AI Measurement Science

AI Measurement Science

A textbook on the foundations and practice of AI evaluation

Sang T. Truong, and Sanmi Koyejo

Stanford University
Stanford, CA

© 2025 Stanford University

All rights reserved.

Table of Contents

Preface 8
Structure of this book . . . 9
How to engage with this book . . . 11
Prior knowledge . . . 11
Citation . . . 11
1 Validity 13
1.1 A Network of Constructs . . . 15
1.2 A Taxonomy of Validity Evidence . . . 16
1.3 Threats to Validity in AI Evaluation . . . 21
1.4 Bibliographic Notes . . . 23
1.5 Exercises . . . 24
2 Data 25
2.1 The Response Matrix . . . 25
2.2 A Taxonomy of AI Benchmarks . . . 27
2.3 Multilingual and Cultural Evaluation . . . 41
2.4 Preference Data and Pairwise Comparisons . . . 43
2.5 Paired Response Matrices . . . 44
2.6 Gaps in the Landscape . . . 49
2.7 Data Quality in Practice . . . 55
2.8 The `torch_measure` Toolkit . . . 56
2.9 Discussion Questions . . . 58
2.10 Bibliographic Notes . . . 59
3 Models 61
3.1 What Does It Mean to Measure? . . . 62
3.2 Additive Conjoint Measurement . . . 65
3.3 Sufficiency and Rasch Model . . . 67
3.4 Reinterpreting the Second Conjoint Factor . . . 69
3.5 Testing for an Interval Scale . . . 71
3.6 The n-Parameter Logistic Model . . . 76
3.7 Exercises . . . 79
3.8 Bibliographic Notes . . . 83
4 Learning 85
4.1 Why Learning Matters for AI Measurement . . . 86
4.2 Maximum Likelihood Estimation . . . 87
4.3 Joint, Conditional, and Marginal MLE . . . 93
4.4 The EM Algorithm . . . 95

4.5 Bayesian Inference . . . 100
4.6 Regularization and Model Selection . . . 107
4.7 Discussion Questions . . . 109
4.8 Bibliographic Notes . . . 110
4.9 Exercises . . . 111
5 Reliability 113
5.1 Sources of Variability in AI Measurement . . . 114
5.2 From the Response Model to Variance Components . . . 118
5.3 Variance Estimation via Plug-in Method . . . 120
5.4 Variance Estimation via Bayesian Method . . . 124
5.5 Variance Estimation via Method of Moments . . . 126
5.6 Design for Generalizability . . . 132
5.7 Model Selection: Which Facets to Control . . . 134
5.8 Generalizability for Factor Models . . . 136
5.9 Generalizability for Ordinal Response . . . 139
5.10 Generalizability for CTT . . . 147
5.11 Item-Level Diagnostics . . . 150
5.12 Building Trustworthy AI Evaluations . . . 162
5.13 Design principles . . . 167
5.14 Bibliographic Notes . . . 168
5.15 Exercises . . . 169
6 Efficiency 173
6.1 The Design Problem in AI Evaluation . . . 174
6.2 Optimal Experimental Design . . . 174
6.3 Discussion Questions . . . 186
6.4 Bibliographic Notes . . . 186
6.5 Exercises . . . 186
7 Discovery 188
7.1 How Many Constructs? Dimensionality Assessment . . . 189
7.2 The Factor Model . . . 192
7.3 Reading What Each Item Measures: The Structure Matrix . . . 194
7.4 Item Clustering and Benchmark Heterogeneity . . . 195
7.5 Generalization of the Factor Model . . . 196
7.6 When There Is No Common Cause: Network Models . . . 200
7.7 Exercises . . . 204
7.8 Bibliographic Notes . . . 205
8 Predictability 206
8.1 Amortized Models . . . 206
9 Intervention 208
9.1 From Association to Causation in Benchmark Scores . . . 209
9.2 Interventional and Counterfactual Diagnostics . . . 211
9.3 Distribution Shift and Transportability . . . 213
9.4 Off-Policy Evaluation and Doubly Robust Estimation . . . 217
9.5 Conformal Prediction Under Distribution Shift . . . 225

9.6 Putting It All Together: A Causal Audit . . . 229
9.7 Discussion Questions . . . 230
9.8 Bibliographic Notes . . . 230
9.9 Exercises . . . 231
10 Ecosystem 233
10.1 From Scoreboard to Target . . . 234
10.2 The Actors and Their Incentives . . . 234
10.3 A Taxonomy of Gaming . . . 237
10.4 The Gaming-Channel / Design-Lever Map . . . 238
10.5 The Governance Frontier . . . 239
10.6 Discussion Questions . . . 240
10.7 Bibliographic Notes . . . 240
10.8 Exercises . . . 241
11 Design 242
11.1 When Measurement Becomes a Target . . . 243
11.2 What Are We Trying to Hide? . . . 244
11.3 The Evaluation Game . . . 247
11.4 Hiding the Set . . . 248
11.5 Hiding the Distribution . . . 251
11.6 Publishing the Construct: The Construct-Validity Protocol . . . 264
11.7 Metric Design as Principal-Agent Problem . . . 265
11.8 Representative Benchmark Selection . . . 269
11.9 Synthesis: Design Principles for Strategic Benchmarks . . . 274
11.10 Discussion Questions . . . 275
11.11 Bibliographic Notes . . . 276
11.12 Exercises . . . 278
12 Application 280
12.1 Red-Teaming as Measurement . . . 281
12.2 When Can Attack Success Rates Be Compared? . . . 284
12.3 Adversarial Robustness Evaluation . . . 290
12.4 Synthetic Data for Evaluation . . . 294
12.5 Evaluation at Scale . . . 298
12.6 Putting It Together: An Adversarial Evaluation Pipeline . . . 302
12.7 Discussion Questions . . . 303
12.8 Bibliographic Notes . . . 303
12.9 Exercises . . . 304
13 Conclusion 306
13.1 Our Approach . . . 306
13.2 Lessons from the Field . . . 307
13.3 Open Challenges . . . 309
13.4 Capstone Projects . . . 310
13.5 A Practitioner's Checklist . . . 313
13.6 Scope and Limitations . . . 315
13.7 Final Thought . . . 315

Appendices 317
Notation 317
General . . . 317
Item Response Theory . . . 317
Learning and Estimation . . . 318
Reliability . . . 319
Validity . . . 319
Causality and Distribution Shift . . . 320
Information and Mechanism Design . . . 320
Red-Teaming and Adversarial Evaluation . . . 322
References 323
Index 340

Preface

A textbook on the foundations and practice of AI evaluation

Decisions about AI systems—whether to deploy one, which to prefer, how to govern its use—rest on measurements of quantities such as reasoning, planning, and safety. The quality of those measurements is therefore a precondition for sound decisions, yet it is rarely examined. New benchmarks appear faster than their measurement properties can be characterized; leaderboards report scores without specifying the scale on which they are defined; and a system's average accuracy is largely uninformative about why it succeeds or where it fails. The field has accumulated a large number of evaluation results, but comparatively little theory governing how those results should be produced and interpreted.

This book develops the foundations and practical tools of measurement science for AI evaluation. Its organizing premise is that evaluation is a problem of inference - where we estimate latent properties of a system, such as ability along a given dimension, from observed responses, - rather than a procedure that ends once a dataset is collected and a metric is computed. From this premise the book builds outward. It begins with validity, the question of whether an evaluation measures the construct it claims to, and with the structure of benchmark data. It then develops the probabilistic models that make latent properties estimable — item response theory, Bradley–Terry models, and factor models — together with methods for fitting them, for decomposing measurement error, and for designing evaluations that extract more information per item. On this base it treats the harder questions: when benchmark results generalize across shifts in distribution, and how to evaluate systems that are themselves optimizing against the benchmark, which brings in causal inference and mechanism design. Throughout, each method is presented with its assumptions, the conditions under which it applies, and the ways it can fail, since characterizing the failure modes of an evaluation is as much a part of measurement as the estimate itself.

The book is intended for researchers, practitioners, and students who work on or care about AI evaluation. We assume some background in probability, statistics, and machine learning, but provide sufficient context and references for readers to follow the main ideas. The book also includes interactive code examples and datasets. The science of AI measurement is a fast-moving area with many open challenges, and we hope this book will inspire readers to further explore and advance it. We hope with the present book to help the field move from ad hoc benchmarking toward principled measurement—so that our decision about AI systems rest on solid scientific ground.

Stanford, 2026, Truong & Koyejo

Structure of this book

The chapters build in sequence — from validity and the structure of benchmark data, through the models, estimation, reliability, and design that form the methodological core, to causal generalization and the strategic dimensions that arise once benchmarks become optimization targets.

Chapter 1 opens the book with validity—the question of whether an evaluation measures the construct it claims to. The chapter models a construct as a latent variable embedded in a nomological network that causally produces item responses, and adopts existence-and-causation condition as the standard validity must meet. A central difficulty is that this network is not identifiable from data alone—substantively different structures can imply the same observed responses—so validity cannot be read off the data and must instead be built up as an accumulation of evidence. The chapter organizes that evidence, following the unitary view of validity, into five kinds (content, criterion, construct, external, and consequential), and catalogs the major threats at the conceptual level, deferring their operationalization to the later chapters.

Chapter 2 surveys the AI evaluation data landscape: the response matrix abstraction, a taxonomy of 70+ benchmarks spanning knowledge, coding, agentic, multilingual, cultural, and preference evaluation, and the practical data quality issues that arise in real-world evaluation.

Chapter 3 lays the mathematical modeling groundwork for the rest of the book. It covers probabilistic models for AI evaluation data — including Item Response Theory (Rasch, 2PL, 3PL) and Bradley-Terry models for pairwise comparisons. A central theme is the relationship between model assumptions and what they allow us to infer: sufficiency, specific objectivity, and the tradeoffs between model complexity and interpretability. The chapter also introduces prediction-powered evaluation—an amortized model that maps external features to latent parameters for cold-start prediction of unseen models and items.

Chapter 4 studies how to estimate latent parameters from evaluation data. It covers maximum likelihood estimation, gradient-based optimization, the EM algorithm, Bayesian inference, regularization, and model selection for the unidimensional IRT models.

Chapter 5 addresses the question of signal versus noise in AI measurement. Building on the response model of Chapter 3, it derives a variance decomposition—person, item, interaction, and residual—and defines reliability as the person share of that variance, the generalizability coefficient. It develops three routes for estimating the variance components (plug-in, Bayesian, and method of moments), turns generalizability theory into a design tool that sizes items and raters to a target reliability, and treats the rater facet in depth for LLM-as-a-judge and ordinal-rating protocols—including why systematic biases such as a judge's position bias can be reliable yet escape reliability analysis. Classical test theory appears as the special case recovered when only total scores are available. It is also where the validity threats of Chapter 1 become operational: the item-level diagnostics—differential item functioning, dimensionality assessment, and item-fit statistics—are each a structured reading of the same variance decomposition.

Chapter 8 turns from confirming a known construct to discovering an unknown one. Where the preceding chapters assume the construct is named in advance and ask whether the items measure it, this chapter is handed a collection of items and asks the data how many distinct things they measure and which items measure which. It uses parallel analysis to recover dimensionality, then develops the multidimensional factor model—its definition, estimation, the structure matrix and item clustering for diagnosing benchmark heterogeneity, and masking experiments that test how well the learned model generalizes to unseen models and items. It closes with network models (the Gaussian graphical model and the Ising model) that drop the common-cause assumption entirely—treating item correlations as direct associations rather than the signature of a shared latent ability—and shows why latent-variable and network structures can be observationally close.

Chapter 6 develops the statistical foundations for efficient evaluation design. It introduces Fisher information for item selection, Computerized Adaptive Testing (CAT), D-optimal design for constructing maximally informative item pools, and efficient paired-comparison schedules for model arenas.

Chapter 9 examines the causal foundations of AI measurement. It covers structural causal models for evaluation data, distribution shift (covariate, label, and concept shift) and when benchmark results generalize across contexts, interventional and counterfactual reasoning for diagnosing construct-irrelevant variance, and methods for robust prediction under shift including conformal inference and data attribution.

Chapter 10 steps back from the statistics to the institutions that consume benchmark scores. It maps the stakeholders of a modern benchmark—builders, evaluators and platforms, adopters, investors, and regulators—and argues that as a benchmark's audience expands, Goodhart's Law turns from an academic curiosity into a structural force, analyzed through the economics of signaling and the market for lemons. It develops a mechanism-based taxonomy of gaming on both sides of the market—builders who game the metric and evaluators who game the audience—and connects each gaming channel to a design lever, framing the governance frontier (multi-principal evaluation, the role of open-source models, and how to organize trustworthy third-party evaluation). The chapter is the diagnosis that motivates the design machinery of Chapter 11.

Chapter 11 considers the strategic and economic dimensions of evaluation design. It introduces decision-making under ambiguity (Maxmin Expected Utility), Bayesian persuasion for optimal information disclosure, robust mechanism design, and the strategic considerations that arise when benchmark results influence development incentives.

Chapter 12 applies the measurement framework to adversarial evaluation: red-teaming as structured measurement, attack success rate as an estimand (and when comparisons are valid), adversarial robustness as a multidimensional latent trait, synthetic data for evaluation at scale, and prediction-powered inference for combining human and automated judgments.

Chapter 13 synthesizes the book's themes, distills lessons from the field, identifies six open challenges (beyond binary responses, multidimensional ability, temporal dynamics, agentic

evaluation, scalable oversight, and fairness in evaluation), presents ten capstone projects for the CS321M course, and offers a practitioner's checklist for designing rigorous AI evaluations.

How to engage with this book

Three chapters are essential for every reader—Chapter 1, Chapter 2, and Chapter 3—which frame the measurement question, the data landscape, and the modeling toolkit that the rest of the book builds on; they belong in both reading tracks. Beyond this shared core, the book splits into two tracks. Readers focused on building and improving rigorous evaluations should follow the methodological track—Chapter 4, Chapter 5, Chapter 6, and Chapter 9—which develops estimation, reliability, efficient design, and the analysis of when results generalize. Readers focused on the strategic and governance dimensions of AI evaluation should follow Chapter 10, Chapter 11, and Chapter 12, which cover the incentives surrounding benchmarks, the mechanism design that responds to them, and adversarial testing.

Prior knowledge

The book assumes knowledge of the fundamentals of statistics, linear algebra, and machine learning. Code examples are written in Python using PyTorch, scikit-learn, and pandas, so familiarity with Python is valuable for readers who wish to engage with the interactive examples. For readers seeking to strengthen their prerequisites or deepen their understanding of the fields this book draws from, we recommend the following companion texts:

- Borsboom (2005) — A philosophically rigorous account of what it means to measure a psychological attribute. Essential reading for the validity and construct modeling discussions in Chapter 5 and Chapter 1.
- Lord and Novick (1968) — The classical statistical theory of mental testing that underlies much of modern psychometrics. For readers wanting depth in reliability, item analysis, and test construction before the measurement chapters.
- Murphy (2022) — A modern, comprehensive treatment of probabilistic modeling, Bayesian inference, and statistical learning. Excellent preparation for the latent variable models and estimation methods in Chapter 3 through Chapter 6.

Citation

To cite this book, please use the following entry:

```
@book{truongkoyejo2026aims,
  title     = {AI Measurement Science},
  author    = {Truong, Sang T. and Koyejo, Sanmi},
  year      = {2026},
  url       = {https://aimsbook.org/}
```

```
6 }
```

1 Validity

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Frame** validity as a question of generalization: distinguish generalization from a sample to its own data distribution (the familiar machine-learning sense) from generalization from benchmark items to the intended construct, and explain why validity concerns the latter.
2. **Define** validity and explain why it cannot be reduced to a single statistic, separating what validity *is*—the causal, existence-and-causation standard—from how it is *established*—the modern unitary view that accumulates evidence for an intended interpretation.
3. **Explain** why a construct's nomological network is underdetermined by observational data (non-identifiability), and why validation is therefore a process of assembling evidence rather than a quantity read off the scores.
4. **Distinguish** content, criterion, construct, external, and consequential validity, and identify what each contributes to the case for an intended interpretation.
5. **Identify** the major threats to validity in AI evaluation—benchmark contamination, construct-irrelevant variance, construct underrepresentation, and differential item functioning—and locate where each is operationally diagnosed later in the book (item-level diagnostics in Section 5.11, generalization failures in Chapter 9).
6. **Evaluate** the validity of existing AI benchmarks using the frameworks presented in this chapter.

Suggested Lecture Plan

This chapter can be covered in **1 lecture** (75–90 minutes):

- From scores to claims: validity as a generalization question, and why it is prior (15 min)
- The nomological network and Borsboom's realist condition; non-identifiability (20 min)
- A taxonomy of validity evidence: content, criterion, construct, external, consequential (30 min)
- Threats to validity in AI evaluation, and where each is diagnosed later in the book (25 min)

A benchmark typically reports a number about specific items: a model answered 430 of 500 questions, won 62% of its matches, passed 18 of 20 held-out tests. But we almost never care about those particular items; we report the number because we expect it to *generalize*. The sense of generalization familiar from machine learning is from a finite sample to the distribution it was drawn from—test accuracy as an estimate of expected performance on more items of the same kind. The generalization a benchmark is actually used for is more ambitious. When a score is used—to deploy a model, to rank one lab above another, to certify a system as safe—it is read as a statement about a hidden property of the system: reasoning, coding ability, factual reliability,

safety. Such a property—a general attribute we cannot observe directly but try to measure through items that depend on it—is called a *construct*, and it is the central object of this chapter. To read "passed 430 of 500 questions" as "has strong reasoning ability" is to make a larger leap: to claim the score generalizes not merely to more items drawn the same way, but to the construct itself—to differently worded items, other task formats, and the deployment conditions we ultimately care about. *Validity* is the name for whether that generalization holds.

This kind of generalization is logically prior to the statistical questions that occupy the rest of this book. Larger item pools, tighter confidence intervals, and better estimators all sharpen the first, narrower kind of generalization—they reduce the error in estimating performance on the benchmark's own distribution—but they say nothing about the second. If performance on the benchmark distribution does not generalize to the construct, a more precise estimate of that performance is of no help.

The leap can fail even when the numbers look impeccable. Consider a coding benchmark whose scores are highly stable—run it twice and the rankings barely move, and every internal consistency check comes back clean. One would naturally conclude that the benchmark is working well. But suppose the items were scraped from a popular programming-tutorial site, and the top-scoring models turn out to be exactly those whose training data included it. The benchmark is not measuring *coding ability*—it is measuring *memorization of specific solutions*. The scores are real and reproducible, yet the inference they invite, "these models can code," is false. A benchmark can be flawless by every internal numerical criterion and still license the wrong claim.

It helps to separate what validity *is* from how we *establish* it. As a definition, we adopt the causal standard: an evaluation is valid for a construct when that construct exists and variation in it causally produces variation in the scores. This is a sharp criterion—the scraped coding benchmark fails it because the variation in scores is caused by memorization, not by coding ability—but it is not one we can read directly off the data, since many different causal structures can produce the same observed responses. So in practice we operationalize validity epistemically, following the modern unitary view (Messick 1995): *the degree to which evidence and theory support the intended interpretation of a test's scores*. On this view validity is a property of the interpretation we attach to the scores, not of the test in isolation, and establishing it is a matter of assembling evidence for the causal claim the definition demands.

The rest of this chapter builds the apparatus for doing so. We first make the target precise by modeling a construct and its relationship to the items as a *nomological network* (Section 1.1), which states the causal structure to be tested and makes clear why it is underdetermined by data; we then organize the evidence that bears on an intended interpretation into five kinds (Section 1.2); and we catalog the characteristic ways the item-to-construct inference breaks down in AI evaluation (Section 1.3). The chapter is deliberately conceptual: the tools for operationally *detecting* these failures build on machinery developed later, so we return to them as item-level diagnostics (Section 5.11) and principles for building trustworthy evaluations (Section 5.12).

1.1 A Network of Constructs

A construct does not exist in isolation. It acquires meaning through its place in a *nomological network*—a web of lawful relationships connecting a construct to its observable measures and to the downstream outcomes it should predict (Cronbach and Meehl 1955). Throughout this chapter we model constructs as latent variables that causally produce the measures (a single benchmark may probe more than one construct at once). Drawn out, the network is then a graphical model over the constructs, their measurements, and the outcomes they should predict (Frank et al. 2025), and validity becomes the question of whether the relationships we *observe* are consistent with the structure the network *asserts* (Figure 1.1). From a causal perspective, Denny Borsboom provides a definition for validity. A measurement instrument is valid for measuring a construct if and only if:

1. The construct theoretically exists, and
2. Variations in the construct causally produce variations in the measurement response.

First, the construct being measured must exist independent of the measurement itself. If we claim to measure "reasoning" but there is no such thing—if intelligence is better understood as a collection of independent abilities—then no test can validly measure it. Second, the construct must cause variation in measurement responses. It is not enough for test scores to be correlated with the construct; the construct must be the reason for the variation.

A natural next step is to instantiate the construct within the nomological network. One compelling ontological hypothesis is that the construct is the common cause of the responses. The common cause can then be represented as a latent variable in a graphical model. This is the main idea behind latent variable models, such as Item Response Theory, where test responses are generated by a latent ability. The ability is a property of the test taker that exists independently of any particular test and causally produces the responses.



READING GRAPHICAL MODELS

The diagrams in this chapter are *graphical models*: nodes are variables, and a directed edge $A \to B$ means A is among the causes that generate B. Following the standard shading convention, **open circles** are *latent* (unobserved) variables—a construct θ, or an item's unique factor z_j—and **shaded circles** are *observed* variables, such as a response Y_j. Edges are read generatively, from tail to head: the construct produces the responses, not the other way around. Chapter 3 extends this notation with *plates*—rectangles that compactly denote replication over many models and items—once we turn to models that estimate these edges.

Reading the network from left to right gives this chapter its structure:

- An **intervention** I acts on the causes of a construct θ—in AI, the training data, objective, supervised fine-tuning, or RL. This is the handle we use to test whether θ genuinely produces the responses (Chapter 9).
- Each **construct** θ causally produces the scores Y_j on the items that load on it, and an item may load on more than one construct (like Y_2). This $\theta \to Y$ pathway is the core of validity.

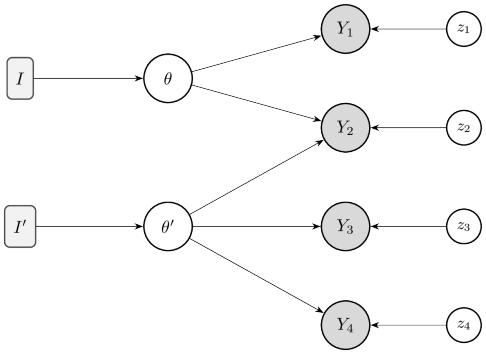


Figure 1.1
The nomological network as a graphical (measurement) model. Open circles are latent, shaded circles are observed. Two latent constructs θ, θ' generate four benchmark items: Y_1 loads on θ alone, Y_2 cross-loads on both constructs, and Y_3, Y_4 load on θ' alone. Each item also carries an item-specific latent variable (unique factor) z_j, capturing variance not shared with the constructs. An intervention (I on θ, I' on θ') acts on the *causes* of each construct (e.g., training data, objective, supervised fine-tuning, reinforcement learning)—the AI-native handle for probing whether a construct causally produces the responses. The loading edges $\theta \to Y$ are the subject of construct validity; items sharing a construct should converge while items on different constructs should discriminate; and the z_j are construct-irrelevant, item-specific variance.

  - Each **item** also carries an item-specific latent variable z_j: variance unique to that item and not shared with any construct—a source of construct-irrelevant variance.

The constructs ultimately matter because they predict downstream outcomes we care about, such as deployment usefulness; relating scores to those outcomes is the province of criterion and predictive validity (Section 1.2.2).

1.2 A Taxonomy of Validity Evidence

The problem of the nomological network is that it is **not identifiable** from observational data alone: genuinely different networks can produce exactly the same observed data. Figure 1.2 shows a multi-construct example. Four items split into two abilities, θ (measured by Y_1, Y_2) and θ' (measured by Y_3, Y_4), whose scores are positively correlated. Panel (a) explains that correlation with two *correlated* factors. Panel (b) explains it with a **bifactor** model: a general factor g that loads on all four items, together with orthogonal *specific* factors s_1, s_2 for each pair. These are substantively different theories of the construct space—"two related abilities" versus "one general ability running through everything, plus residual specifics"—yet they can imply the *identical* covariance over $Y_1, \ldots, Y_4$, and nothing in the observed responses says which is right.

This is the classic ambiguity of the general factor. A positive correlation among abilities is explained equally well by "the abilities are simply related" (panel a) and by "a general ability runs through them all" (panel b); the bifactor even has *extra* freedom, so it fits at least as well by construction. The covariance can establish *that* θ and θ' covary; it cannot establish *why*. The same indistinguishability afflicts the meaning of any single construct: whether a benchmark's shared variance reflects the ability we intend or a construct-irrelevant cause such as memorized

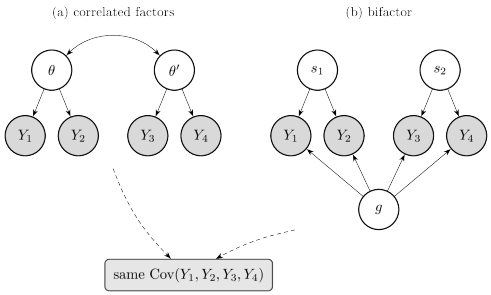


Figure 1.2
Two multi-construct latent networks that can imply the same observable data. Four items load on two abilities, θ (Y_1, Y_2) and θ' (Y_3, Y_4). (a) Correlated-factors model: the two abilities simply covary. (b) Bifactor model: a general factor g loads on every item, with orthogonal specific factors s_1, s_2. For suitable parameters both reproduce the same Cov($Y_1, \dots, Y_4$)—indeed a proportionally constrained bifactor is mathematically the correlated two-factor model—so the data alone cannot tell us whether a general factor exists.

text—the scraped coding benchmark from the start of this chapter is exactly this trap. Whenever we commit to one network, we are choosing among structures the data cannot separate.

The nomological network thus gives us a *definition* of validity, but because the structure is underdetermined, it does not let us simply read validity off the data. Validation is instead a process of accumulating evidence. The evidence-accumulation tradition we adopt as our workflow—Cronbach, Messick, Kane, and the claim-centered view—answers an *epistemic* question: how do we come to know that a test supports its intended interpretation?

The modern view of validity, following Messick (1995) and the Standards for Educational and Psychological Testing, treats validity as a *unitary concept*: the degree to which evidence supports the intended interpretation of test scores. However, the evidence for validity comes in several distinguishable forms. Following Salaudeen et al. (2025) and the classical framework, we organize validity evidence into five categories.

1.2.1 Content Validity

Content validity concerns whether the benchmark adequately *represents* the construct domain it claims to measure. The key questions are: Does the benchmark cover the full range of the construct? Are the items relevant? Are important aspects of the construct missing? For AI evaluation, content validity failures are pervasive. A benchmark labeled "reasoning" might contain only arithmetic word problems, missing logical reasoning, causal reasoning, analogical reasoning, and spatial reasoning entirely. A "coding ability" benchmark might test only Python function completion, neglecting debugging, system design, documentation, and code review. Content validity is established through *expert judgment* rather than statistical analysis. Domain experts review the construct definition, examine the item pool, and assess coverage. In AI evaluation, this step is frequently skipped: benchmarks are constructed from convenience samples of existing data rather than from principled domain specifications.

AI example. Suppose a benchmark claims to measure "scientific reasoning." Content validity requires asking: Does it include hypothesis generation? Experimental design? Data interpreta-

tion? Statistical inference? Causal reasoning? If it tests only factual recall of scientific knowledge, it has poor content validity for the "reasoning" claim, regardless of how reliable the scores are.

1.2.2 Criterion Validity

Criterion validity asks whether benchmark scores predict or correlate with an external criterion that independently captures the construct. There are two subtypes:

- **Concurrent validity**: the benchmark correlates with a *currently available* gold standard. For example, an automated coding benchmark should correlate with expert human evaluation of the same code.
- **Predictive validity**: the benchmark predicts *future outcomes* of interest. A benchmark for coding assistants should predict how useful the assistant is in actual developer workflows over time.

Criterion validity provides some of the most compelling evidence, but it requires the existence of a trustworthy external criterion—which is often the fundamental problem. If we already had a perfect measure of "reasoning ability," we would not need the benchmark. The circularity of criterion validation is a recognized challenge in psychometrics (Cronbach and Meehl 1955) and becomes especially acute in AI evaluation, where the construct of interest (e.g., "general intelligence") may not have any agreed-upon gold standard.

AI example. The Chatbot Arena (Zheng et al. 2023) collects pairwise human preferences as a criterion for model quality. A benchmark has concurrent criterion validity if its scores correlate with Arena Elo ratings. But the Arena itself has validity assumptions—are crowdworker preferences a valid criterion for "model quality"?

1.2.3 Construct Validity

Construct validity is the central and most encompassing form of validity evidence. It asks: does the benchmark actually measure the *theoretical construct* it claims to measure? This goes beyond content coverage and criterion correlation to the internal structure of the measurement. The classical approach to construct validity uses two kinds of evidence:

- **Convergent validity**: scores on the benchmark should correlate *positively* with scores on other measures of the same or closely related constructs. If two different "reasoning" benchmarks produce uncorrelated scores, at least one of them has poor construct validity.
- **Discriminant validity**: scores on the benchmark should *not* correlate strongly with measures of *different* constructs. If a "reasoning" benchmark correlates as highly with a "memorization" benchmark as with another reasoning benchmark, the construct is not well separated.

The systematic study of convergent and discriminant validity was formalized by Campbell and Fiske (1959) through the **Multitrait-Multimethod (MTMM) matrix**, which we develop in Section 5.11.5.

Under Borsboom's causal framework, construct validity requires a specific causal claim: variation in the latent construct (reasoning ability) causally produces variation in benchmark scores. This is testable—if we could intervene to increase a model's reasoning ability while holding everything else constant, valid benchmark scores should increase. In practice, such clean interventions are rare, but the causal framing clarifies what we are asking.

AI example. Consider two benchmarks that both claim to measure "mathematical reasoning": GSM8K (grade-school math word problems) and MATH (competition-level problems). If they measure the same construct, models that excel on one should tend to excel on the other (convergent validity). If "mathematical reasoning" is distinct from "commonsense knowledge," then GSM8K scores should correlate less with commonsense benchmarks than with MATH (discriminant validity).

1.2.4 External Validity

External validity concerns the *generalizability* of benchmark results beyond the specific conditions of the evaluation. Does performance on the benchmark predict performance in other contexts, with other populations, or at other times?

For AI systems, external validity questions include:

- Does performance on English-language benchmarks predict multilingual performance?
- Do results on curated, clean test items generalize to noisy, real-world inputs?
- Do benchmark scores from today predict performance after the model is updated?
- Do results generalize across deployment contexts (e.g., from research settings to production)?

External validity connects closely to the problem of *distribution shift*, which we examine formally in Chapter 9. When the benchmark distribution differs from the target distribution, strong benchmark performance may not transfer.

AI example. A model scores 95% on a medical question-answering benchmark derived from textbook questions. But in a clinical setting, the questions are noisier, contextually embedded, and may involve ambiguous information. The benchmark has poor external validity if the 95% score does not predict clinically useful performance.

1.2.5 Consequential Validity

Consequential validity, introduced by Messick (1995), asks whether the *social consequences* of using benchmark scores are appropriate. This is the most controversial form of validity evidence, as it extends validity beyond measurement science into ethics and policy. In AI evaluation, consequential validity is increasingly important:

- **Goodhart's law**: When a benchmark becomes a target, models are optimized for it, and the benchmark ceases to measure the original construct. Training on benchmark-like data improves scores without improving capability.
- **Development distortion**: Benchmarks that are easy to optimize attract disproportionate effort, even if they measure less important capabilities.
- **Misuse of rankings**: Leaderboard positions are used to make deployment decisions, marketing claims, and policy arguments that go far beyond what the scores support.

Consequential validity does not mean that benchmarks are "invalid" whenever they have negative consequences. Rather, it means that the consequences of score use are *relevant evidence* when evaluating whether the measurement system is serving its intended purpose.

AI example. A safety benchmark becomes widely used for regulatory compliance. Model developers learn to optimize specifically for the benchmark items, achieving high scores while leaving genuine safety risks unaddressed. The consequential validity of the benchmark is undermined: the scores are being used to support claims ("this model is safe") that they do not actually support.

1.2.6 Summary

Table 1.1 summarizes the five forms of validity evidence, their key questions, and how they manifest in AI evaluation.

Table 1.1
Five forms of validity evidence for AI evaluation

Form	Key Question	AI Example	Typical Evidence
Content	Does the benchmark cover the construct domain?	"Reasoning" benchmark tests only arithmetic	Expert review, domain specification
Criterion	Do scores predict an external criterion?	Benchmark vs. Arena Elo correlation	Correlation with gold standard
Construct	Does the benchmark measure the intended construct?	Convergent/discriminant evidence across benchmarks	MTMM matrix, factor analysis
External	Do results generalize beyond test conditions?	English benchmark → multilingual performance	Cross-context replication
Consequential	Are the social consequences appropriate?	Goodhart's law on safety benchmarks	Impact analysis, misuse audit

1.3 Threats to Validity in AI Evaluation

Even well-designed benchmarks face systematic threats that can undermine the validity of the inferences they support. We identify four major categories.

1.3.1 Construct-Irrelevant Variance

Construct-irrelevant variance (CIV) occurs when systematic factors *other than the target construct* influence benchmark scores. Unlike random noise (which reduces reliability), CIV is systematic and can inflate or deflate scores in predictable ways.

Sources of CIV in AI evaluation include:

- **Prompt formatting sensitivity**: The same question presented with different formatting (bullet points vs. paragraphs, numbered vs. lettered options) can change model responses substantially (Mizrahi et al. 2024). If the construct is "reasoning," but scores depend on formatting, the format contributes construct-irrelevant variance.
- **Multiple-choice position bias**: Many language models show systematic preferences for certain answer positions (e.g., option A or the last option). This inflates scores for items where the correct answer happens to be in the preferred position.
- **Tokenization artifacts**: Model performance can depend on how text is tokenized, which is an artifact of the model's vocabulary rather than its ability.
- **Language and cultural bias**: Items that assume specific cultural knowledge or linguistic patterns may be easier for models trained predominantly on certain data, independent of the target construct.

CIV is particularly insidious because it is *systematic*: unlike random error, it does not average out with more items. A benchmark full of items with position bias has high reliability (the bias is consistent) but poor validity (the scores reflect position preference, not just the target ability).

Graphically, systematic CIV is an *additional parent* of the items, sitting alongside the construct (Figure 1.3). Writing the response of model i to item j as

$$Y_{ij} = \underbrace{\lambda_j \theta_i}_{\text{construct}} + \underbrace{\gamma_j N_i}_{\text{construct-irrelevant}} + \underbrace{z_{ij}}_{\text{item-specific}},$$

the nuisance cause N (formatting sensitivity, position preference, ...) loads on *several* items through the γ_j. This sharing across items is what makes CIV *systematic*: a cause confined to a single item would collapse into that item's unique factor z_{ij} and merely add noise, whereas a cause spread across items acts as a second common factor and biases the construct estimate. The practical consequence turns on whether N is observed. If N is *observed*—answer position, prompt length, language—it can be controlled for or randomized away (counterbalancing position decorrelates it from the score). If N is *latent*, it is confounded with θ: a single benchmark cannot separate the two, and the multitrait-multimethod design of Section 5.11.5 is precisely the tool for doing so.

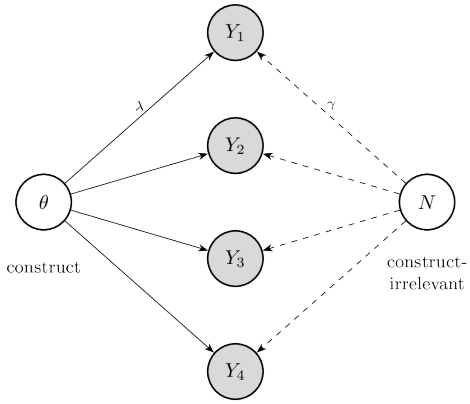


Figure 1.3
Systematic construct-irrelevant variance as a graphical model. The target construct θ (left, solid loadings λ) and a construct-irrelevant cause N (right, dashed loadings γ) are *both* parents of the observed items. N is a validity threat—rather than mere item noise—because it loads on several items at once; a cause acting on a single item would be absorbed into that item's unique factor z_j. When N is latent it is confounded with θ, and a single test cannot tell their contributions apart.

1.3.2 Construct Underrepresentation

Construct underrepresentation occurs when the benchmark is *too narrow*, sampling only a limited aspect of the construct it claims to measure. This is the content validity threat from Section 1.2.1 viewed through a different lens.

Examples in AI evaluation:

- **Coding ability** tested only through function-completion tasks, missing debugging, architecture design, code review, documentation, and refactoring.
- **Language understanding** tested only with formal, well-edited text, missing colloquial language, code-switching, and domain-specific jargon.
- **Safety** tested only through adversarial prompts in English, missing multilingual attacks, multi-turn manipulation, and system-prompt override attempts.

Construct underrepresentation is often a consequence of convenience sampling: benchmarks are built from data that is easy to collect and annotate, not from a principled specification of the construct domain.

1.3.3 Benchmark Contamination

Benchmark contamination occurs when evaluation items appear in the model's training data, allowing the model to retrieve memorized answers rather than demonstrating the target capability. This is arguably the most discussed validity threat in current AI evaluation.

Contamination can be *direct* (exact match between training and test items) or *indirect* (paraphrases, derivatives, or items from the same source that share structural patterns). Detection approaches include:

- **Canary strings**: Embedding unique identifiers in benchmark items and testing whether models can reproduce them (Jacovi et al. 2023).
- **Membership inference**: Statistical tests for whether a model has seen specific items during training.
- **Chronological splits**: Comparing performance on items created before vs. after the model's training data cutoff.
- **Public/private splits**: Items released publicly tend to have inflated scores compared to held-out private items—the gap measures contamination effects.
- **Performance discontinuities**: If a model's accuracy on "seen" items is dramatically higher than on matched "unseen" items, contamination is likely.

Under Borsboom's causal framework, contamination is a validity threat because it introduces an alternative causal path: the model's response is caused by *memory* rather than by the target *ability*. The score no longer reflects the construct it claims to measure.

1.3.4 Differential Item Functioning

A fourth threat is **differential item functioning (DIF)**: an item is systematically easier or harder for one subgroup of test-takers than another *after controlling for overall ability*. In AI evaluation the "groups" are typically model families, architectures, or training paradigms, and a DIF item rewards a construct-irrelevant feature—a formatting convention, a tokenization quirk—rather than the target ability, making it a group-structured special case of the construct-irrelevant variance of Section 1.3.1. Detecting DIF is an operational task: it reuses the same variance decomposition as reliability (a DIF term is an item-by-group interaction), so we develop its machinery—Mantel-Haenszel, logistic-regression, and IRT-based tests—alongside that decomposition in Section 5.11.1.

1.4 Bibliographic Notes

The conceptual foundations of validity have evolved substantially over the past century. Cronbach and Meehl (1955) introduced the notion of construct validity and the nomological network, arguing that validity requires embedding a construct in a web of theoretical relationships. Messick (1995) unified the previously separate types of validity into a single framework, arguing that all validity is construct validity, with content, criterion, and consequential evidence as *facets* of a unified concept. Kane (2006) formalized the argument-based approach to validation, which requires specifying the chain of inferences from observed scores to intended interpretations and gathering evidence for each link.

Borsboom (2005) challenged the orthodox framework with a realist account: a test is valid if and only if the target construct exists and causally produces score variation. This causal perspective connects validity to the structural causal models discussed in Chapter 9.

For AI-specific validity, Salaudeen et al. (2025) propose a claim-centered framework that adapts classical validity concepts for benchmark evaluation. Kiela et al. (2021) argue for

dynamic benchmarking to combat contamination and Goodhart's law. Jacovi et al. (2023) address the contamination problem directly, proposing practical strategies for protecting test data. The broader challenges of AI evaluation validity are discussed by Shankar et al. (2024) and Biderman et al. (2024). S. T. Truong et al. (2025) demonstrate that item-level validity threats — incorrect answer keys, ambiguous wording, grading bugs — can be detected at scale using reliability diagnostics derived from the Rasch model's sufficiency property. Their framework achieves up to 84% precision at the top-50 flagged items across nine benchmarks, illustrating how measurement theory yields operational tools for benchmark quality assurance.

The relationship between reliability and validity was previewed in Chapter 5, where we established that reliability is a necessary but not sufficient condition for validity. The causal aspects of validity—particularly the relationship between Borsboom's causal definition and structural causal models—are developed further in Chapter 9.

1.5 Exercises

1. Using Borsboom's causal definition of validity, argue whether "prompt sensitivity" (a model's score changes when the prompt template is varied, holding the question content constant) is a *validity* threat or a *reliability* threat. Under what conditions might it be both?
2. A coding benchmark produces scores with test-retest reliability of 0.95 and Cronbach's alpha of 0.93. Is this sufficient evidence for validity? What additional evidence would be needed, and why?
3. How does benchmark contamination differ from construct-irrelevant variance? Describe a scenario where training data overlap is actually *construct-relevant* rather than a validity threat.
4. Goodhart's law suggests that any benchmark used as an optimization target will eventually lose validity. Does this make consequential validity fundamentally different from the other four forms? Or is it a special case of construct-irrelevant variance?
5. DIF analysis (developed in Section 5.11.1) identifies items that function differently across groups after controlling for ability. In AI evaluation, when should DIF be treated as a validity problem to be fixed, and when might it reflect a *genuine* group difference that the benchmark should capture?
6. Salaudeen et al. (2025) argue that validity should be assessed relative to specific claims rather than as a global property of the benchmark. Borsboom (2005) argues that validity is fundamentally about whether the target construct exists and causally produces score variation. Compare these two frameworks. Under what conditions do they agree? Under what conditions might they disagree? Which framework is more useful for guiding practical benchmark design, and why?

2 Data

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Describe** the response matrix Y_{ij} as the universal data structure underlying AI evaluation and identify what constitutes a "subject" and an "item" across different evaluation paradigms.
2. **Classify** AI benchmarks along multiple axes: domain, response type, evaluation structure, and cultural scope.
3. **Articulate** how design choices in benchmark construction — item selection, scoring rubrics, response format — shape the resulting response matrix and constrain downstream analysis.
4. **Identify** practical data quality issues in AI evaluation: sparsity, missing data, inconsistent scoring, and benchmark contamination.
5. **Use** the `torch_measure` toolkit to load, inspect, and visualize response matrices from real benchmarks.

Suggested Lecture Plan

This chapter can be covered in **1 lecture** (75–90 minutes):

- The response matrix abstraction (15 min)
- A taxonomy of AI benchmarks (20 min)
- Multilingual, multicultural, and domain-specific evaluation (15 min)
- Preference data and pairwise comparisons (10 min)
- Data quality and practical issues (10 min)
- Hands-on: loading and exploring benchmarks with `torch_measure` (15 min)

2.1 The Response Matrix

Every AI evaluation, no matter how complex, ultimately produces a table: rows are the systems being evaluated (models, agents, model–scaffold combinations), columns are the evaluation items (questions, tasks, prompts), and entries record how each system performed on each item. This is the **response matrix**.

Definition: Response Matrix

A **response matrix** $Y \in \mathbb{R}^{N \times M}$ records the performance of N subjects (models) on M items (tasks). The entry Y_{ij} can be:

- **Binary**: $Y_{ij} \in \{0, 1\}$ (correct/incorrect, pass/fail, resolved/unresolved)
- **Continuous**: $Y_{ij} \in [0, 1]$ (pass@1 rate, partial credit score, token probability)
- **Missing**: $Y_{ij} =$ `NaN` (model not evaluated on this item)

The response matrix is the fundamental data structure for measurement science. All of the models in this book — IRT, factor models, Bradley-Terry — operate on response matrices or transformations thereof.

The simplicity of this abstraction is deceptive. The same $N \times M$ matrix structure accommodates radically different evaluation paradigms:

Evaluation Type	Rows (Subjects)	Columns (Items)	Values	Example
Knowledge QA	LLMs	Multiple-choice questions	Binary	MMLU-Pro (48 × 12,257) (Y. Wang et al. 2024)
Code generation	LLMs	Programming problems	Binary or pass@1	BigCodeBench (153 × 1,140) (Zhuo et al. 2025)
Agent tasks	Agent + scaffold	Multi-step episodes	Binary	SWE-bench (134 × 500) (Jimenez et al. 2023)
Function calling	LLMs	API call specifications	Binary	BFCL (93 × 4,751) (Patil et al. 2025)
Web navigation	Agent + scaffold	Web interaction tasks	Binary	WebArena (14 × 812) (Zhou et al. 2024)
Terminal tasks	Agent + scaffold	System admin tasks	Continuous	Terminal-Bench (128 × 89) (Merrill et al. 2026)

Code reasoning	LLMs	Input/output prediction	Continuous	CRUXEval (38 × 800) (Gu et al. 2024)

The key design decision is what counts as a "subject." For knowledge benchmarks, a subject is typically a single LLM. For agentic benchmarks, a subject is a model–scaffold combination (e.g., SWE-Agent + Claude Sonnet 4), because the scaffold's search strategy, tool use, and error recovery contribute substantially to performance. This distinction matters for measurement: if the scaffold contributes variance, a model's "ability" as estimated by IRT partially reflects the scaffold, not the model alone.

2.1.1 Shape and Sparsity

Response matrices in AI evaluation are often surprisingly sparse or oddly shaped. A benchmark with 500 items evaluated on 134 model–scaffold combinations (SWE-bench Verified (Jimenez et al. 2023)) produces a dense matrix. But a benchmark ecosystem where models are evaluated on different subsets of items — because benchmarks evolve over time, or because compute constraints limit which models run on which items — produces a matrix with systematic missing data.

LiveCodeBench (Jain et al. 2024) illustrates this: its 72 models × 1,055 problems matrix is only 88.8% filled, because older models were evaluated on earlier problem sets (713 or 880 problems) while newer models have the full 1,055. This is not missing-at-random — it is missing-by-design, and the missingness pattern carries information (newer models tend to be more capable).

Understanding the shape and sparsity of the response matrix is a prerequisite for choosing the right model. Dense, rectangular matrices support standard IRT and factor models. Sparse or systematically incomplete matrices require models that handle missing data explicitly, or imputation strategies that account for the missingness mechanism.

2.2 A Taxonomy of AI Benchmarks

The AI evaluation landscape has grown rapidly. To make sense of it, we organize benchmarks along four axes: **domain**, **response type**, **evaluation structure**, and **cultural scope**.

2.2.1 By Domain

Benchmarks cluster into broad capability domains, each with distinct item characteristics and validity considerations.

Knowledge and reasoning. Benchmarks like MMLU-Pro (12,257 items across 14 domains) (Y. Wang et al. 2024), LiveBench (White et al. 2025), and HLE (Phan et al. 2026) test

factual knowledge and reasoning through multiple-choice or short-answer questions. Items are typically self-contained, automatically scored, and drawn from existing exams or expert-written question banks. The primary validity concern is construct underrepresentation: a "reasoning" benchmark that tests only factual recall does not measure reasoning.

Code generation and software engineering. This is the largest and most diverse category, spanning basic function completion (EvalPlus: HumanEval+ and MBPP+ (Jiawei Liu et al. 2023)), competitive programming (LiveCodeBench: 1,055 problems from AtCoder, LeetCode, and CodeForces (Jain et al. 2024)), library-aware code generation (BigCodeBench: 1,140 tasks using 139 APIs (Zhuo et al. 2025)), code reasoning (CRUXEval: 800 input/output prediction problems (Gu et al. 2024)), code editing (EditBench: 540 editing tasks (Chi et al. 2026)), and full software engineering (SWE-bench: resolving real GitHub issues (Jimenez et al. 2023)). The progression from function completion to issue resolution represents increasing ecological validity — and increasing difficulty of automated scoring.

Agentic tasks. A rapidly growing category where the "subject" is not a bare model but a model–scaffold combination operating in an interactive environment. Examples include web navigation (WebArena: 812 tasks across e-commerce, forums, and CMS sites (Zhou et al. 2024)), mobile device automation (AndroidWorld: 116 tasks (Rawles et al. 2025)), desktop interaction (OSWorld (T. Xie et al. 2024)), multi-app coordination (AppWorld: 24 tasks (Trivedi et al. 2024)), and terminal operations (Terminal-Bench: 89 system administration tasks (Merrill et al. 2026)). Agentic benchmarks pose unique measurement challenges: the "item" is a multi-step episode, performance depends on the scaffold as much as the model, and scoring may require environment rollback and verification.

Tool use and function calling. BFCL (4,751 items across 22 categories) (Patil et al. 2025) and ToolBench (Xu et al. 2023) test whether models can correctly invoke APIs, parse schemas, and handle multi-turn tool interactions. These benchmarks sit between pure language tasks and agentic tasks — they test a specific capability (structured output generation) rather than end-to-end task completion.

Safety, security, and red teaming. This category spans cybersecurity challenges (CyBench: 40 CTF tasks (A. K. Zhang et al. 2025)), security of tool-using agents (AgentDojo: 949 security items (Debenedetti et al. 2024)), and red teaming — adversarial evaluation of whether models can be induced to produce harmful outputs. Red teaming data has a natural response matrix structure: rows are models, columns are attack prompts, and values are binary (safe/unsafe). HarmBench (400 prompts across 7 harm categories) (Mazeika et al. 2024) provides a standardized red teaming benchmark; BeaverTails (334K prompts with fine-grained safety annotations) (Ji et al. 2023) and DecodingTrust (243K prompts across 8 trustworthiness dimensions) (B. Wang et al. 2024) offer larger-scale evaluation. SafetyBench (11,435 MCQs across 7 safety categories) (Z. Zhang et al. 2024) tests safety knowledge in a traditional MCQ format, while WMDP (3,668 MCQs on biosecurity, cybersecurity, and chemical security) (N. Li et al. 2024) evaluates hazardous knowledge as a proxy for CBRN risk. These benchmarks are distinctive because the "correct" response is often a *refusal*, and the construct (safety) is inherently adversarial — a model that scores perfectly on a safety benchmark today may fail tomorrow against novel attacks.

Preference and reward modeling. Benchmarks like RewardBench (Lambert et al. 2024), AlpacaEval (805 instructions) (X. Li et al. 2023), MT-Bench (Zheng et al. 2023), Arena Hard (Tianle Li* 2024), WildBench (Lin et al. 2024), and the Chatbot Arena (140K+ comparisons) (Chiang et al. 2024) evaluate model quality through human or automated preference judgments. The response matrix structure differs: instead of $Y_{ij} \in \{0, 1\}$, entries may represent win rates, Elo ratings, or pairwise comparison outcomes. We discuss this structure separately in Section 2.4.

2.2.2 By Response Type

The granularity of the response determines which measurement models are appropriate.

Binary responses ($Y_{ij} \in \{0, 1\}$) are the simplest and most common. Standard IRT models (Rasch, 2PL, 3PL) are designed for binary data. Most code generation and knowledge benchmarks use binary scoring: the answer is correct or it is not.

Continuous responses ($Y_{ij} \in [0, 1]$) arise from partial credit scoring, pass@k estimation (where Y_{ij} is the empirical pass rate over k samples), or rubric-based evaluation. Terminal-Bench scores tasks on a 0–100 scale (Merrill et al. 2026); CRUXEval reports pass@1 from 10 samples (Gu et al. 2024). Continuous responses carry richer information than binary responses and motivate extensions like Beta-IRT (Chapter 4).

Ordinal responses ($Y_{ij} \in \{1, 2, \ldots, L\}$) arise from Likert-scale rubrics (e.g., 1–5 quality ratings). The graded response model and partial credit model extend IRT to ordinal data, but these are less common in current AI evaluation.

Preference data ($Y_{ij} \in \{A, B, \text{tie}\}$) from pairwise comparisons have a fundamentally different structure, discussed in Section 2.4.

2.2.3 By Evaluation Structure

Static benchmarks have a fixed item set evaluated once per model. Most existing benchmarks are static. The advantage is reproducibility; the disadvantage is vulnerability to contamination and saturation.

Dynamic benchmarks add new items over time (LiveBench (White et al. 2025), LiveCodeBench (Jain et al. 2024)) or generate items adversarially (DynaBench (Kiela et al. 2021)). LiveCodeBench draws from ongoing programming competitions, ensuring that items postdate model training cutoffs (Jain et al. 2024). The measurement challenge is maintaining scale comparability: if the item pool changes, ability estimates from different time periods are not directly comparable without equating procedures.

Interactive benchmarks require multi-turn interaction between the model and an environment or human evaluator. Chatbot Arena (Chiang et al. 2024), WebArena (Zhou et al. 2024), and Terminal-Bench (Merrill et al. 2026) are interactive. The "item" is not a static question but a dynamic episode whose difficulty depends on the model's earlier actions. Standard IRT assumes

item parameters are fixed and independent of the subject — an assumption that interactive benchmarks may violate.

2.2.4 Complete Benchmark Inventory

The following tables enumerate all 84 benchmarks curated in the `torch_measure` collection, organized by domain and sorted by release date. Together they span over 4.5 million unique evaluation items and over 6,700 model/agent entries, with 69 benchmarks providing full per-item response matrices.

BENCHMARK INVENTORY (70 BENCHMARKS, 1.2M+ ITEMS, 6,700+ TEST-TAKERS)

Knowledge and Reasoning

#	Benchmark	$N \times M$	Type	Evaluation Method	Released	Reference
1	ARC-AGI v1	52 × 400	Bin	Grid output match; visual abstract reasoning	2019-11	arXiv:1911.01547[1]
2	MMLU-Pro	48 × 12,257	Bin	MCQ; exact match on 10-choice questions	2024-06	arXiv:2406.01574[2]
3	LiveBench	195 × 494	Con	Rubric-scored; monthly-refreshed, automated grading	2024-06	arXiv:2406.19314[3]
4	ARC-AGI v2	28 × 120	Bin	Grid output match; harder visual reasoning	2024-12	arcprize[4]
5	HLE	19 × 1,792	Bin	MCQ + open-ended; expert-authored, LLM-graded	2025-01	arXiv:2501.14249[5]
6	MathArena	68 × 336	Con	Exact match; competition problems (AIME, AMC)	2025-03	matharena.ai[6]

Code Generation and Software Engineering

#	Benchmark	$N \times M$	Type	Evaluation Method	Released	Reference
7	EvalPlus	31 × 542	Bin	Unit tests; augmented test suites	2023-05	arXiv:2305.01210[7]
8	SWE-bench Verified	134 × 500	Bin	Repo test suite; GitHub issue resolution	2023-10	arXiv:2310.06770[8]
9	SWE-bench Full	24 × 2,294	Bin	Repo test suite; full instance set	2023-10	arXiv:2310.06770[9]
10	CRUXEval	38 × 800	Con	Exact match; I/O prediction	2024-01	arXiv:2401.03065[10]
11	LiveCodeBench	72 × 1,055	Con	Unit tests; pass@1, contest problems	2024-03	arXiv:2403.07974[11]
12	BigCodeBench	153 × 1,140	Bin	Unit tests; sandbox, 139 library APIs	2024-06	arXiv:2406.15877[12]
13	SWE-bench Java	52 × 170	Bin	Repo test suite; Java issues	2024-08	multi-swe-bench[13]
14	SWE-bench Multi	13 × 301	Bin	Repo test suite; multi-language	2024-08	multi-swe-bench[14]
15	MLE-bench	30 × 75	Con	Kaggle scoring; competition submission	2024-10	arXiv:2410.07095[15]
16	DPAI Arena	9 × 141	Con	Test suite + rubric; dual evaluation	2025-01	dpaia.dev[16]
17	ClineBench	3 × 12	Con	Harbor framework; coding agent	2025-01	cline/cline[17]
18	SWE-PolyBench	— × 2,110	Bin	Repo test suite; polyglot SWE	2025-01	arXiv:2501.14798[18]
19	EditBench	44 × 540	Con	Unit tests; code editing, multilingual	2025-02	waynchi/edit-bench[19]

Tool Use and Function Calling

#	Benchmark	$N \times M$	Type	Evaluation Method	Released	Reference
20	BFCL v3	93 × 4,751	Bin	AST match + exec; function call validation	2024-02	arXiv:2402.15671[20]
21	ToolBench	10 × 765	Bin	StableToolBench; cached API eval	2024-03	arXiv:2403.07714[21]

Agentic Tasks

#	Benchmark	$N \times M$	Type	Evaluation Method	Released	Reference
22	WebArena	14 × 812	Bin	Browser env; live websites	2023-07	arXiv:2307.13854[22]
23	AgentBench	29 × 8	Con	Multi-env; OS, DB, web, game	2023-08	arXiv:2308.03688[23]
24	GAIA	32 × 165	Bin	Exact match; web + tool-use	2023-11	arXiv:2311.12983[24]
25	VisualWebArena	6 × 910	Con	Browser env; multimodal web	2024-01	arXiv:2401.13649[25]
26	WorkArena	4 × 118	Con	ServiceNow env; enterprise	2024-03	arXiv:2403.07718[26]
27	OSWorld	77 × 369	Con	VM env; desktop automation	2024-04	arXiv:2404.07972[27]
28	AndroidWorld	3 × 116	Bin	Emulator; mobile automation	2024-05	arXiv:2405.14573[28]
29	AgentDojo	29 × 132	Bin	Sandbox; tool-use + security	2024-06	arXiv:2406.13352[29]
30	AgentDojo (Sec.)	28 × 949	Bin	Sandbox; attack success	2024-06	arXiv:2406.13352[30]
31	TAU-bench	32 × 329	Con	Simulated env; customer service	2024-06	arXiv:2406.12045[31]
32	AppWorld	18 × 31	Con	API env; multi-app interaction	2024-07	arXiv:2407.18901[32]
33	CORE-Bench	15 × 270	Bin	Docker env; reproducibility	2024-09	arXiv:2409.11353[33]
34	BrowserGym	18 × 8	Con	Browser env; aggregate scores	2024-12	arXiv:2412.05467[34]
35	TheAgentCompany	19 × 175	Con	Simulated enterprise; workplace	2024-12	arXiv:2412.14161[35]
36	Terminal-Bench	128 × 89	Con	Docker env; CLI task resolution	2025-02	arXiv:2502.10996[36]
37	PaperBench	9 × 20	Con	Rubric; reproduce ML papers	2025-02	arXiv:2504.01848[37]

Safety, Security, and Red Teaming

#	Benchmark	$N \times M$	Type	Evaluation Method	Released	Reference
38	CyBench	8 × 40	Bin	CTF env; flag capture	2024-08	arXiv:2408.08926[38]
39	BBQ	7 × 58,492	Bin	MCQ; bias across 11 demographic categories	2022-05	nyu-mll/BBQ[39]
40	JailbreakBench	18 × 100	Bin	Red teaming; 5 attack methods × 4 models	2024-01	JailbreakBench[40]
41	BeaverTails	15 × 33,432	Bin	Safety annotations; 14 harm categories	2023-07	PKU-Alignment/beaver-tails[41]
42	HarmBench	— × 510	—	Classifier judge; 7 harm categories (items only)	2024-02	arXiv:2402.04249[42]
43	WMDP	— × 3,668	—	MCQ; biosecurity, cybersecurity, chemical (items only)	2024-03	wmdp.ai[43]
44	SafetyBench	— × 11,435	—	MCQ; 7 safety categories (items only, answers withheld)	2024-06	thu-coai/Safety-Bench[44]
45	DecodingTrust	— × 243K	—	8 trustworthiness axes (prompts only)	2023-06	NeurIPS 2023[45]
46	TensorTrust	— × 563K	Bin	Prompt injection attacks + defenses (game)	2024-02	qxcv/tensor-trust[46]
47	LLMail-Inject	839 × 40	Bin	Prompt injection; 40 levels, multiple LLMs	2025-06	microsoft/llmail-inject[47]
48	Alignment Faking	— × 2.14M	Bin	RL transcripts; alignment faking labels	2024-12	Anthropic/alignment-faking-rl[48]
49	AgentHarm	— × 176	Bin	Multi-step harmful agent tasks; 11 categories	2024-10	ai-safety-institute/AgentHarm[49]
50	MACHIAVELLI	— × 572K scenes	Con	Ethical decision-making; 25+ violation types	2023-04	aypan17/machiavelli[50]
51	BELLS	— × 5 datasets	Bin	Labeled execution traces; jailbreak, hallucination	2024-06	CeSIA/BELLS[51]
52	Scale MRT	— × 6K traces	Bin	Agent monitor evasion; lying, manipulation	2025	ScaleAI/mrt[52]

Domain-Specific (Legal, Finance, Medical)

#	Benchmark	$N \times M$	Type	Evaluation Method	Released	Reference
53	MedNLI	— × 14,049	Bin	NLI; clinical sentence pairs from MIMIC-III	2018-08	Romanov and Shivade (2018)
54	PubMedQA	— × 1,000	Bin	MCQ; biomedical research yes/no/maybe	2019-09	Jin et al. (2019)
55	IgakuQA	5 × 1,471	Bin	MCQ; Japanese medical licensing exams (2018–2022)	2023-03	jungokasai/IgakuQA[53]
56	LawBench	51 × 9,000	Bin	Exact match; 20 Chinese legal tasks, zero-shot	2023-09	open-compass/LawBench[54]
57	FinanceBench	16 × 150	Bin	Expert-graded; SEC filing QA	2023-11	patronus-ai/financebench[55]
58	LexEval	38 × 14,147	Con	Rubric; 23 Chinese legal tasks (NeurIPS 2024)	2024-09	CSHaitao/LexEval[56]
59	AfriMedQA	30 × 6,910	Bin	MCQ; Pan-African medical, 20 specialties	2024-09	arXiv:2409.15290[57]

Preference and Reward Modeling

#	Benchmark	$N \times M$	Type	Evaluation Method	Released	Reference
79	AlpacaEval	221 × 805	Bin	LLM judge; win/loss vs GPT-4	2023-05	tatsu-lab/alpaca_eval[58]
80	MT-Bench	— × 80	—	GPT-4 + human pairwise; multi-turn	2023-06	lm-sys/FastChat[59]
81	UltraFeedback	17 × 63,966	Con	GPT-4 judge; overall score (1–10)	2023-10	OpenBMB/UltraFeedback[60]
82	RewardBench	149 × 2,985	Bin	Score comparison; chosen vs rejected	2024-03	arXiv:2403.13787[61]
83	WildBench	63 × 1,024	Con	LLM judge; checklist scoring	2024-06	arXiv:2406.04770[62]

Multilingual and Cultural Evaluation

#	Benchmark	$N \times M$	Type	Evaluation Method	Released	Reference
84	MasakhaNER v2	— × —	Bin	NER; 20 African languages	2022-10	arXiv:2210.12391[63]
78	HELM African	— × —	Con	HELM harness; African language tasks	2022-11	HELM[64]
79	HELM CLEVA	— × —	Con	HELM harness; Chinese language eval	2022-11	HELM[65]
80	HELM ThaiExam	— × —	Con	HELM harness; Thai examination tasks	2022-11	HELM[66]
81	C-Eval	— × 12,342	Bin	MCQ; Chinese educational system	2023-05	arXiv:2305.08322[67]
82	CMMLU	— × 11,582	Bin	MCQ; Chinese 67 subjects	2023-06	arXiv:2306.09212[68]
83	Rakuda	141 × 40	Con	LLM judge; Japanese open-ended QA	2023-06	shisa-ai/shaberi[69]
84	SIB-200	2 × 41,820	Bin	Classification; 205 languages	2023-09	arXiv:2309.07445[70]
78	OALL Arabic MMLU	— × —	Bin	MCQ; native Arabic knowledge questions	2024-02	OALL[71]
79	OALL Arabic Exams	— × —	Bin	MCQ; Arabic exam questions	2024-02	OALL[72]
80	KMMLU	— × 35,030	Bin	MCQ; Korean exams	2024-02	arXiv:2402.11548[73]
81	Tengu-Bench	141 × 120	Con	LLM judge; Japanese multi-category	2024-04	shisa-ai/shaberi[74]
82	AfriEval	— × 469,210	Bin	MCQ + NLI + QA; African languages	2024	HuggingFace
83	AsiaEval	— × 398,263	Bin	MCQ + NLI; Asian languages	2024	HuggingFace
84	CulturalEval	— × 82,996	Bin	MCQ; cross-cultural values	2024	HuggingFace
78	IberBench	— × 32,797	Bin	MCQ + NLI; Iberian languages	2024	HuggingFace
79	Bridging Gap	1,767 × 36	Bin	MCQ; Winogrande × 12 languages	2024	HuggingFace
80	Ko Leaderboard	1,159 × 9	Con	lm-eval-harness; Korean tasks	2024	Open Ko-LLM[75]
81	La Leaderboard	69 × 108	Con	lm-eval-harness; Iberian tasks	2024	HuggingFace
82	PT Leaderboard	1,148 × 10	Con	lm-eval-harness; Portuguese	2024	HuggingFace
83	Thai Leaderboard	72 × 19	Con	lm-eval-harness; Thai tasks	2024	HuggingFace
84	TUMLU	30 × 7,486	Bin	MCQ; 9 Turkic languages, CoT + non-CoT	2024-12	ceferisbarov/TUMLU[76]

Legend. N = models/agents; M = items/tasks; "—" = item-only data. **Type**: Bin = binary; Con = continuous $[0, 1]$. **Released**: earliest public availability (arXiv, GitHub, or HuggingFace). **Evaluation Method**: MCQ = multiple-choice exact match; unit tests = code execution; LLM judge = automated preference judgment; env = interactive environment.

Scoring variants. The 84 entries above correspond to 120 dataset variants on HuggingFace (`aims-foundation/torch-measure-data`). Many benchmarks are released with multiple **scoring conditions** that produce different response matrices from the same items — a measurement choice that affects downstream analysis. For example:

Benchmark	Variants	Scoring conditions
CRUXEval	6	Continuous / binary × combined / input-only / output-only
TAU-bench	6	Combined + 5 domain splits (airline v1/v2, HAL, retail, telecom)
EvalPlus	5	Combined + HumanEval / MBPP × base / augmented test suites
BigCodeBench	4	Complete / instruct × full / hard subset
DPAI Arena	4	Total / blind / informed scoring + binary threshold
MLE-bench	4	Continuous / binary / above-median / raw Kaggle scores
CyBench	3	Unguided / subtask-guided / subtask completion scores
SWE-PolyBench	2	Full (1 model) / verified (3 models)
13 others	2 each	Binary vs. continuous rescoring of same items

These variants represent different *testing conditions* in the sense of Chapter 5: the same items administered under different scoring rubrics, prompt formats, or subset selections. A model's measured ability can change substantially across conditions — BigCodeBench "instruct" scores differ from "complete" scores for the same model on the same items, because the prompt format changes what capability is being measured. This is why Generalizability Theory (Chapter 5) decomposes variance across conditions: the scoring condition is a facet of measurement, not just a data processing choice.

2.2.5 Visualizing the Landscape

To see the structure of the evaluation landscape at a glance, we embed the item text from 19 benchmarks using a sentence transformer, then project benchmark centroids to 2D with UMAP.

Each point represents one benchmark, positioned by the semantic content of its items.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: landscape-scatter
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

# Pre-computed UMAP coordinates (sentence-transformer embeddings of sampled items)
data = {"x": [9.83, 10.16, 10.43, 8.61, 8.97, 8.29, 9.37, 8.64, 8.7, 10.74, 11.07,
 ↪  10.46, 7.17, 7.24, 6.42, 6.6, 9.24, 11.09, 11.24], "y": [4.02, 3.49, 4.35, 2.86,
 ↪  2.23, 2.27, 1.11, 1.55, 0.85, 6.12, 5.26, 5.53, 9.26, 8.68, 9.42, 8.89, 0.36,
 ↪  3.78, 4.64], "category": ["Knowledge", "Knowledge", "Knowledge", "Knowledge
 ↪  (Chinese)", "Knowledge (Chinese)", "Knowledge (Korean)", "Knowledge (African)",
 ↪  "Knowledge (Asian)", "Knowledge (Iberian)", "Code Generation", "Code Generation",
 ↪  "Code Reasoning", "Software Engineering", "Software Engineering", "Software
 ↪  Engineering", "Software Engineering", "Software Engineering", "ML Engineering",
 ↪  "Tool Use"], "benchmark": ["MMLU-Pro", "HLE", "LiveBench", "C-Eval", "CMMLU",
 ↪  "KMMLU", "AfriEval", "AsiaEval", "IberBench", "BigCodeBench", "EvalPlus",
 ↪  "CRUXEval", "SWE-bench", "SWE-bench Full", "SWE-bench Java", "SWE-bench ML",
 ↪  "SWE-PolyBench", "MLE-bench", "BFCL"]}

# Color map by coarse category
cat_colors = {
    "Knowledge": "#5B8DEE",
    "Knowledge (Chinese)": "#45BF7C",
    "Knowledge (Korean)": "#45BF7C",
    "Knowledge (African)": "#B07CD8",
    "Knowledge (Asian)": "#B07CD8",
    "Knowledge (Iberian)": "#B07CD8",
```

```
    "Code Generation": "#F0A35C",
    "Code Reasoning": "#F0A35C",
    "Software Engineering": "#E8637A",
    "ML Engineering": "#F0A35C",
    "Tool Use": "#888888",
}
# Coarse legend labels
coarse_map = {
    "Knowledge": "Knowledge (English)",
    "Knowledge (Chinese)": "Knowledge (CJK)",
    "Knowledge (Korean)": "Knowledge (CJK)",
    "Knowledge (African)": "Knowledge (Other Lang)",
    "Knowledge (Asian)": "Knowledge (Other Lang)",
    "Knowledge (Iberian)": "Knowledge (Other Lang)",
    "Code Generation": "Code",
    "Code Reasoning": "Code",
    "Software Engineering": "Software Engineering",
    "ML Engineering": "Code",
    "Tool Use": "Tool Use",
}

fig, ax = plt.subplots(figsize=(6, 4))

# Plot points
for i in range(len(data["x"])):
    cat = data["category"][i]
    ax.scatter(data["x"][i], data["y"][i], c=cat_colors[cat], s=60,
               edgecolors="white", linewidths=0.5, zorder=3)
    # Label
    offset = (5, 5)
    if data["benchmark"][i] in ("SWE-bench Full", "SWE-bench Java"):
        offset = (5, -10)
    elif data["benchmark"][i] == "CMMLU":
        offset = (-35, -12)
    elif data["benchmark"][i] == "EvalPlus":
        offset = (5, -10)
    ax.annotate(data["benchmark"][i], (data["x"][i], data["y"][i]),
                fontsize=5.5, xytext=offset, textcoords="offset points",
                color="#333333")

# Legend (coarse categories)
from matplotlib.lines import Line2D
legend_items = {
    "Knowledge (English)": "#5B8DEE",
    "Knowledge (CJK)": "#45BF7C",
    "Knowledge (Other Lang)": "#B07CD8",
    "Code": "#F0A35C",
    "Software Engineering": "#E8637A",
    "Tool Use": "#888888",
```

```
}
handles = [Line2D([0], [0], marker='o', color='w', markerfacecolor=c,
                  markersize=6, label=l) for l, c in legend_items.items()]
ax.legend(handles=handles, loc="lower left", framealpha=0.9)

ax.set_xlabel("UMAP 1")
ax.set_ylabel("UMAP 2")
ax.set_title("AI Evaluation Landscape (benchmark centroids)")
ax.grid(True, alpha=0.2)
plt.tight_layout()
plt.show()

print("19 benchmarks from torch_measure, embedded with all-MiniLM-L6-v2, projected
 ↪  with UMAP.")
print("Knowledge benchmarks (blue/green/purple) cluster by language family.")
print("Code benchmarks (orange) form a distinct region.")
print("SWE-bench variants (red) cluster tightly - similar issue descriptions across
 ↪  languages.")
```

The scatter plot reveals several structural features of the evaluation landscape. **Knowledge benchmarks** form a broad cluster, with English-language benchmarks (MMLU-Pro (Y. Wang et al. 2024), HLE (Phan et al. 2026), LiveBench (White et al. 2025)) grouping together and multilingual benchmarks (C-Eval (Huang et al. 2023), CMMLU (H. Li et al. 2023), KMMLU (Son et al. 2024), AfriEval, IberBench (González et al. 2025)) spreading along a language axis. **Code benchmarks** (BigCodeBench (Zhuo et al. 2025), EvalPlus (Jiawei Liu et al. 2023), CRUXEval (Gu et al. 2024)) occupy a distinct region, reflecting the semantic difference between natural language questions and programming tasks. **Software engineering benchmarks** (SWE-bench (Jimenez et al. 2023) and variants) cluster tightly because their items are GitHub issue descriptions, which share a distinctive technical register regardless of the programming language. The separation between clusters suggests that these benchmark categories measure genuinely different constructs — a hypothesis we can test formally with the structure-matrix and clustering methods of Chapter 8.

2.3 Multilingual and Cultural Evaluation

A striking feature of the current evaluation landscape is the effort to extend measurement beyond English and Western cultural contexts. This creates both opportunities and challenges for measurement science.

2.3.1 Regional Leaderboards and Benchmarks

Multiple evaluation efforts target specific linguistic and cultural communities:

Benchmark	Focus	Coverage
C-Eval (Huang et al. 2023), CMMLU (H. Li et al. 2023)	Chinese language and culture	Mandarin, Chinese educational system
KMMLU	Korean language and knowledge	Korean educational and professional domains
Thai Leaderboard	Thai language evaluation	Thai language tasks
IberBench	Iberian languages	Spanish, Portuguese, Catalan, and related languages
AfriEval	African languages	Multiple African languages and cultural contexts
AfriMedQA	African healthcare	Medical QA in African healthcare contexts
AsiaEval	Asian languages and culture	Cross-Asian evaluation
CulturalEval	Cultural knowledge	Cross-cultural knowledge and values
SIB-200	Massively multilingual	200+ languages, topic classification
HELM Multilingual	Multilingual model evaluation	Standardized multilingual evaluation

These benchmarks reveal a fundamental validity question: does "reasoning ability" — or any other construct we measure — mean the same thing across languages and cultures? A model that excels at English-language reasoning may fail in Korean not because it lacks reasoning ability, but because the items embed cultural knowledge (Korean history, legal system, social norms) that is construct-irrelevant for a non-Korean audience but construct-relevant for Korean users.

In measurement theory terms, this is a **differential item functioning** (DIF) problem at the cultural level. Items that are "fair" in one cultural context may be systematically harder or easier in another, not because of ability differences but because of construct-irrelevant cultural loading. The DIF analysis tools developed in Chapter 1 are directly applicable.

2.3.2 Multilingual Software Engineering

The multilingual dimension extends beyond language tasks. SWE-bench Multilingual (Jimenez et al. 2023) and SWE-bench Java (Zan et al. 2024) test software engineering in languages other than Python, revealing that "coding ability" as measured by Python-only benchmarks may not transfer. SWE-PolyBench (Rashid et al. 2025) tests across multiple programming languages simultaneously. These benchmarks provide natural settings for studying the dimensionality of coding ability: is there a single "software engineering" construct, or are Python ability, Java ability, and JavaScript ability partially independent dimensions?

2.4 Preference Data and Pairwise Comparisons

A significant fraction of AI evaluation data comes not from item-level scoring but from **pairwise comparisons**: a human or automated judge compares two model outputs and declares a winner. The Chatbot Arena (140K+ comparisons) (Chiang et al. 2024), AlpacaEval (X. Li et al. 2023), MT-Bench (Zheng et al. 2023), Arena Hard (Tianle Li* 2024), WildBench (Lin et al. 2024), and preference datasets (HH-RLHF (Bai et al. 2022), UltraFeedback (Cui et al. 2024), HelpSteer2 (Z. Wang et al. 2024), Nectar (Zhu et al. 2023), SHP-2 (Askell et al. 2021)) all produce comparison data.

2.4.1 From Comparisons to Response Matrices

Pairwise comparison data has a different structure from standard response matrices. Instead of "model i on item j," we observe "model A preferred over model B on prompt k." The natural data structure is a **comparison tensor** $C_{ABk} \in \{A, B, \text{tie}\}$.

The Bradley-Terry model , Section 3.4 connects this structure to the response matrix framework. Under Bradley-Terry, the probability that model A is preferred over model B is:

$$P(A \succ B) = \frac{\exp(\theta_A)}{\exp(\theta_A) + \exp(\theta_B)} = \sigma(\theta_A - \theta_B)$$

This is formally equivalent to a Rasch model where the "subject" is the comparison pair (A, B), the "item difficulty" is θ_B, and the "ability" is θ_A. The Elo rating system implements online maximum likelihood estimation for this model.

2.4.2 Reward Model Benchmarks

RewardBench (Lambert et al. 2024) and RewardBench 2 (Malik et al. 2025) evaluate *reward models* — the models that score outputs in RLHF pipelines. Here the "subject" is a reward model and the "item" is a (prompt, chosen response, rejected response) triplet. The response is binary: does the reward model assign a higher score to the chosen response? This is a standard response matrix, but the items have rich internal structure (two full-text responses per item) that simple IRT models do not capture.

Preference Dissection (J. Li et al. 2024) and BigGen (Kim et al. 2025) go further, decomposing preference judgments into multiple criteria (helpfulness, safety, coherence, creativity), producing multi-trait response data suitable for multidimensional measurement models.

2.5 Paired Response Matrices

The response matrices above have one matrix per benchmark. But a growing body of AI evaluation produces *paired* matrices: the same (or comparable) subjects respond to items under multiple conditions — typically with and without AI assistance. Deployment RCTs, uplift studies, and human-AI collaboration experiments all produce this structure.

2.5.1 The Paired Response Matrix

These studies share a common data structure that extends the response matrix with a treatment dimension. Where the standard response matrix records:

$$Y_{ij} \in \{0, 1\} \quad \text{(subject } i \text{ on item } j\text{)}$$

the **paired response matrix** records:

$$Y_{ij}^{(t)} \in \{0, 1\} \quad \text{(subject } i \text{ on item } j \text{ under condition } t\text{)}$$

where $t \in \{\text{control}, \text{treatment}\}$ (or multiple treatment arms). The subjects are now *humans* — radiologists, developers, students — and the measurement target is the **causal effect** of AI on human performance, not the AI system's capability in isolation.

This structure connects naturally to many-facet measurement models. In the many-facet Rasch framework, the treatment condition is simply another facet alongside person ability and item difficulty:

$$\log \frac{P(Y_{ij}^{(t)} = 1)}{P(Y_{ij}^{(t)} = 0)} = \theta_i - \beta_j + \tau_t$$

where τ_t is the treatment effect facet. The interaction $\theta_i \times \tau_t$ captures *heterogeneous treatment effects* — the degree to which AI assistance helps some people more than others.

2.5.2 An Inventory of Paired Response Matrices

We curate five publicly available intervention datasets spanning medicine, software engineering, classification, and education. These are the first datasets in the `torch_measure` collection where the subjects are humans rather than AI systems.

Paired Response Matrix Inventory

Dataset	Domain	Subjects	Items	AI System	Conditions	Outcome	Source
Collab-CXR	Radiology	336 radiologists	324 CXR cases	CheXpert (DenseNet121)	4 (image only, +history, +AI, +AI+history)	Diagnostic probability	OSF[1]
METR Early-2025	Coding	16 developers	246 issues	Cursor Pro + Claude 3.5/3.7 Sonnet	{AI allowed, AI disallowed}	Completion time (min)	GitHub[2]
METR Late-2025	Coding	57 developers	1,134 issues	Cursor Pro + Claude 3.5/3.7 Sonnet	{AI allowed, AI disallowed}	Completion time (min)	GitHub[3]
HAIID	Classification	1,125 participants	152 items (5 domains)	Trained classifiers (per-domain)	Pre/post advice x {AI, human label}	Binary correct	GitHub[4]
GenAI Learning	Education (math)	943 students	57 practice + 48 exam	GPT-4 (vanilla + guardrailed tutor)	{control, vanilla GPT, augmented GPT}	Binary score	GitHub[5]

Collab-CXR (F. Yu et al. 2024) is the largest and cleanest dataset. 227 radiologists each read a subset of 324 chest X-ray cases under four conditions (with/without AI predictions, with/without clinical history), providing probabilistic assessments for 104 pathologies per case. The AI system is **CheXpert**, a DenseNet121 CNN trained on 224,316 chest radiographs that outputs per-pathology probability predictions; it outperforms roughly two-thirds of participating radiologists on AUROC. The within-subject, crossed design makes it ideal for many-facet analysis. AI assistance improves mean diagnostic accuracy from 96.5% to 97.1%, but the effect is heterogeneous across radiologists and pathologies.

METR Developer Productivity (Becker et al. 2025) is a within-subject RCT where each developer's issues are randomly assigned to AI-allowed or AI-disallowed conditions. The AI tool is **Cursor Pro** with **Claude 3.5/3.7 Sonnet** — a state-of-the-art AI coding assistant at the time of the study. The early study (16 developers, 246 tasks) found that AI-allowed tasks took *longer* (119.5 vs. 90.9 minutes) — a counterintuitive result driven by context-switching costs and AI-induced scope expansion. The late study (57 developers, 1,134 tasks) found a smaller and reversed effect (151 vs. 169 minutes), though with severe selection effects. A key limitation for IRT analysis: items are developer-specific (not shared across subjects), producing a block-diagonal matrix.

HAIID (Vodrahalli et al. 2022) measures whether labeling advice as "from AI" versus "from a human" changes how people use it. The AI systems are **trained classifiers** for each domain (art style, city population, sarcasm detection, census income, dermatology). Across five classification domains, the label has minimal effect: AI-labeled advice improves accuracy by +5–9%, and

human-labeled advice improves accuracy by nearly the same amount. The advice *content*, not its provenance, drives the effect.

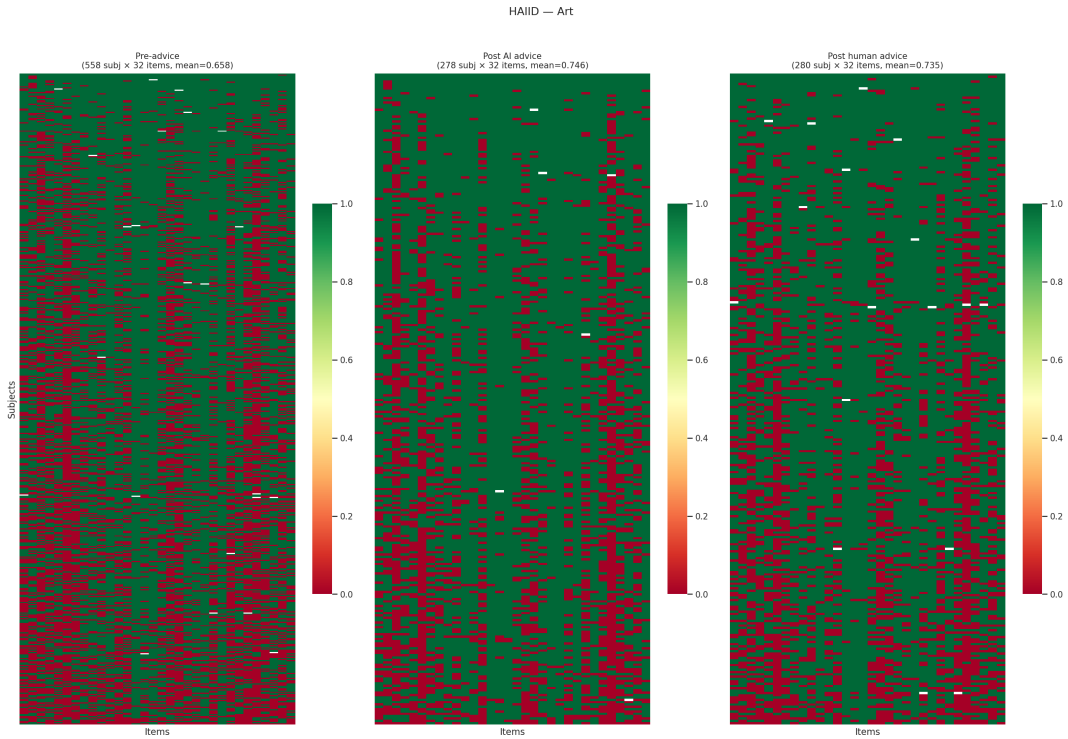


Figure 2.1
Paired response matrices from the HAIID art classification domain. Each panel shows a participant × item response matrix (green = correct, red = incorrect) under a different condition. **Left**: Pre-advice responses (558 participants × 32 items, mean accuracy 0.658). **Center**: Post-advice responses where advice was labeled as "from AI" (278 participants, accuracy 0.746). **Right**: Post-advice responses where advice was labeled as "from a human" (280 participants, accuracy 0.735). The near-identical density of green cells in the center and right panels illustrates that the advice source label has minimal effect on human behavior.

GenAI Learning (Bastani et al. 2025) demonstrates a measurement paradox. The AI system is **GPT-4** in two configurations: a vanilla ChatGPT interface and a pedagogically designed tutor with guardrails that encourages step-by-step reasoning rather than giving answers directly. Students practicing math with ChatGPT score dramatically higher during practice (0.69 vs. 0.34 for controls), but score *no better* on a subsequent exam without AI (0.36 vs. 0.36). Students using the guardrailed GPT tutor also fail to outperform controls on the exam (0.35). AI inflates apparent performance without producing durable learning — a finding with implications for any evaluation that measures human-AI teams without separating the human's contribution.

2.5.3 What Paired Response Matrices Enable

Intervention matrices open measurement questions that standard benchmarks cannot address:

1. **Heterogeneous treatment effects**. Does AI help all users equally, or does it preferentially help experts, novices, or specific subgroups? Many-facet Rasch models and DIF analysis

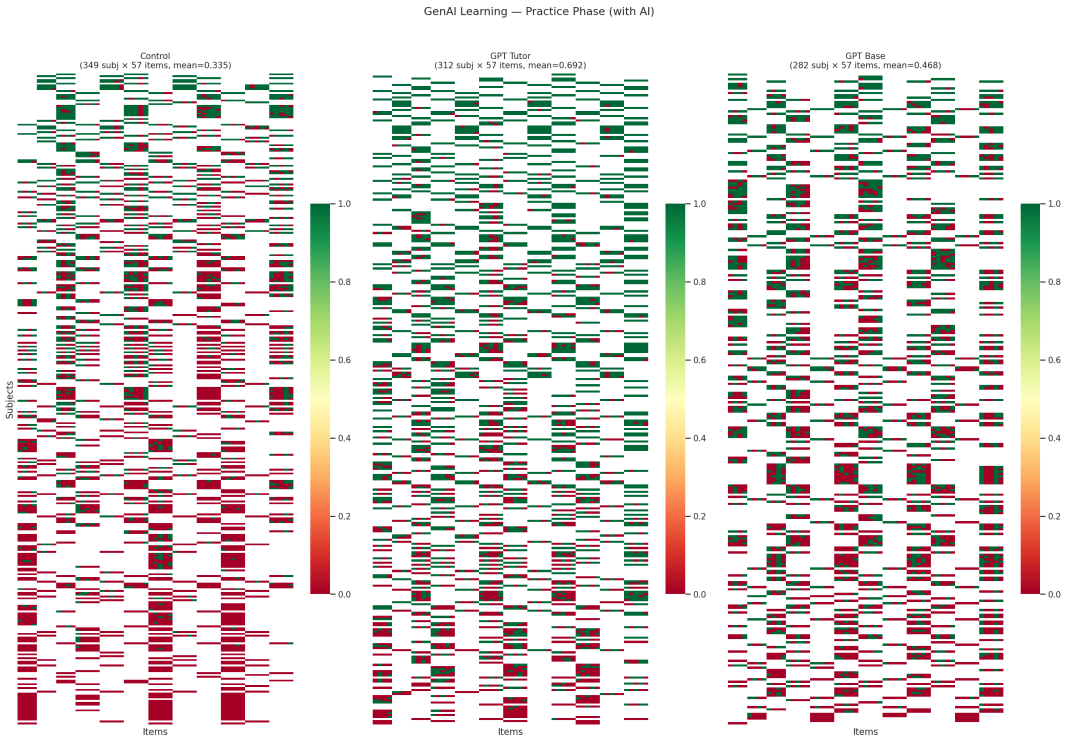


Figure 2.2
Paired response matrices from the GenAI Learning study, practice phase. Each panel shows a student × problem response matrix (green = correct, red = incorrect) under a different treatment arm. **Left**: Control group (349 students, mean 0.335). **Center**: GPT Tutor with pedagogical guardrails (312 students, mean 0.692). **Right**: Vanilla GPT Base (282 students, mean 0.468). The dramatically higher density of green cells in the GPT Tutor panel reflects AI-inflated performance during practice.

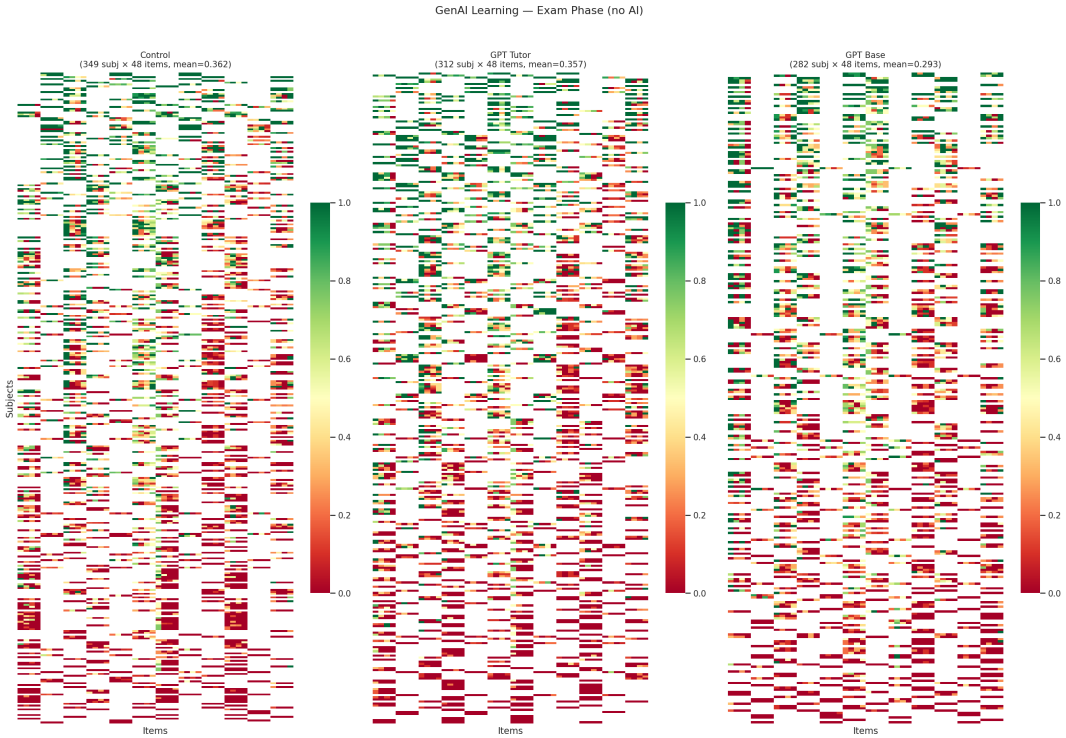


Figure 2.3
The same three treatment arms on a subsequent exam taken *without AI assistance*. **Left**: Control (mean 0.362). **Center**: GPT Tutor (mean 0.357). **Right**: GPT Base (mean 0.293). The visual similarity of the control and GPT Tutor panels — and the increased red in the GPT Base panel — shows that AI-inflated practice performance did not transfer to durable learning.

(Section 5.11.1) can decompose the treatment effect by person characteristics.

2. **Measurement validity of human-AI teams**. If a human-AI team scores 90% on a task, how much is the human contributing? The GenAI Learning result shows this is not a trivial decomposition — apparent team performance can be entirely attributable to the AI, with the human learning nothing.
3. **Linking pre-deployment to post-deployment**. Safety benchmarks measure AI capabilities in isolation; paired response matrices measure what happens when humans interact with those capabilities. The relationship between the two is the empirical question of *ecological validity* — and it is almost entirely unstudied.
4. **Psychometric linking across safety frameworks**. Different AI labs use incompatible safety evaluation frameworks (Anthropic's ASL levels, OpenAI's Preparedness thresholds, DeepMind's Critical Capability Levels). IRT linking methodology could place these on a common scale, but only if item-level data from uplift studies becomes available. Currently, no CBRN uplift study releases item-level data.

2.6 Gaps in the Landscape

The benchmark inventory above is extensive, but the broader landscape is far richer. Understanding what exists *outside* our current collection — and what is genuinely missing — helps identify both curation opportunities and true blind spots.

2.6.1 Domain Coverage

The `torch_measure` collection now spans multiple specialized domains beyond general knowledge and coding. In each domain, we note what is curated, what exists but is not yet curated, and where genuine gaps remain.

Legal. LawBench (51 models × 9,000 Chinese legal items) (Fei et al. 2023) is now curated with full per-question per-model predictions. LegalBench (162 English legal reasoning tasks, 12K+ items) (Guha et al. 2023), LEXTREME (24 European languages) (Niklaus et al. 2023), and CaseHOLD (53K US case law items) (Zheng et al. 2021) have public question datasets but no cross-model prediction matrices. Legal evaluation poses unique measurement challenges: jurisdictional dependence, defensible alternative answers, and multi-faceted constructs spanning statutory interpretation, case analysis, and procedural knowledge.

Finance. FinanceBench (16 model configurations × 150 SEC filing questions) (Islam et al. 2023) is now curated. FinBen/PIXIU (Q. Xie et al. 2024; Xie et al. 2023) covers 36 datasets across 24 tasks using the `lm-evaluation-harness` framework. FinQA (8,281 numerical reasoning questions) (Chen, Chen, et al. 2022), ConvFinQA (multi-turn financial QA) (Chen, Li, et al. 2022), and TAT-QA (hybrid tabular + textual QA) (Zhu et al. 2021) have public datasets but no cross-model prediction data.

Healthcare. AfriMedQA (Pan-African, 30 models × 6,910 items) (Nimo et al. 2025) is curated. MedQA (12,723 USMLE-style questions) (Jin et al. 2020), MedMCQA (194K Indian medical exam questions) (Pal et al. 2022), PubMedQA [1,000 expert-labeled biomedical research questions; Jin et al. (2019)], and MedNLI [14K clinical NLI pairs derived from MIMIC-III; Romanov and Shivade (2018)] all have public question datasets. Multilingual medical benchmarks include CMB/CMExam (Chinese) (X. Wang et al. 2024; Junling Liu et al. 2023), KorMedMCQA (Korean) (Kweon et al. 2024), JMedBench/IgakuQA (Japanese) (Jiang et al. 2025; Kasai et al. 2023), MedExpQA (4 languages) (Alonso et al. 2024), and IMB (Italian, 25K+ items) (Romano et al. 2025). The main barrier is that most leaderboards publish only aggregate accuracy, not per-model per-item response matrices.

Preference and reward modeling. RewardBench (149 reward models × 2,985 items) (Lambert et al. 2024), UltraFeedback (17 models × 64K prompts) (Cui et al. 2024), AlpacaEval (X. Li et al. 2023), and WildBench (Lin et al. 2024) are curated. MT-Bench (80 questions) (Zheng et al. 2023) has item content but only pairwise human judgments, not per-model absolute scores. The Chatbot Arena (33K conversations with pairwise preferences) (Chiang et al. 2024) is available on HuggingFace for pairwise analysis.

Vision-language and multimodal. The OpenVLMRecords dataset on HuggingFace provides per-question raw model responses for 220+ vision-language models across 80+ benchmarks (MMBench (Liu et al. 2024), SEED-Bench (B. Li et al. 2024), MME (Fu et al. 2023), POPE (Y. Li et al. 2023), MM-Vet (W. Yu et al. 2024), MMMU (Yue et al. 2024), MathVista (Lu et al. 2024), Video-MME (Fu et al. 2025), and many more). This is effectively a massive collection of response matrices for multimodal evaluation, ready for IRT analysis. The `lmms-eval` framework (K. Zhang et al. 2025) produces per-item predictions that could be curated similarly. Our collection's limited multimodal coverage (only VisualWebArena (Koh et al. 2024)) is a curation gap, not a data availability gap.

2.6.2 Missing Languages and Cultures

The multilingual benchmark landscape is richer than what our collection currently covers:

Region	In our collection	Per-item data available (not yet curated)	Questions only (no per-model predictions)
East Asia	C-Eval, CMMLU, KMMLU, Thai LB, HELM CLEVA/ThaiExam	Rakuda (~557 models), Tengu-Bench (~558 models), IgakuQA (5 models) — via Shaberi[1]	JGLUE, JMMLU (7,536 MCQs)

[1] https://github.com/shisa-ai/shaberi

Region	In our collection	Per-item data available (not yet curated)	Questions only (no per-model predictions)
Africa	AfriEval, AfriMedQA, HELM African, MasakhaNER v2, Bridging the Gap, SIB-200	—	— (good coverage)
Middle East	OALL Arabic Exams, OALL Arabic MMLU	—	AlGhafa (22,977), ACVA, ORCA
South Asia	Partial via SIB-200, AsiaEval	—	MILU (79,617 MCQs, 11 Indic langs, gated)
Turkic	—	TUMLU (14 models × 38K items × 9 languages, full JSON)	TurkishMMLU (10K)
Eastern Europe	—	—	MERA (19,739 Russian, submissions private), Russian SuperGLUE (101K, private), LLMzSzL (19K Polish), ZNO (3,814 Ukrainian)
Latin America	IberBench, PT LB, La LB	—	— (mostly classification)
Indigenous	—	—	AmericasNLI (10 languages, 14K NLI items)

Three patterns emerge. First, several benchmarks with per-item per-model data exist but are not yet curated in our collection: **TUMLU** (Isbarov et al. 2025) (9 Turkic languages, 14 models, structured JSON with question + model output per item), **Rakuda** (Passaglia 2023) and **Tengu-Bench** (Lightblue KK. 2024) (Japanese, ~558 models via the Shaberi framework), and **IgakuQA** (Kasai et al. 2023) (Japanese medical, 5 baselines). These are ready for immediate curation.

Second, many benchmarks publish questions but not model predictions. JMMLU (Yin et al. 2024) (Japanese), MILU (Verma et al. 2024) (11 Indic languages), TurkishMMLU (Yüksel et al. 2024), LLMzSzL (Jassem et al. 2025) (Polish), and ZNO (Syromiatnikov et al. 2024) (Ukrainian) all have public item sets, but would require re-running models through `lm-evaluation-harness` with `--log_samples` to generate per-item response matrices. Some leaderboards

(MERA (Fenogenova et al. 2024), Russian SuperGLUE (Shavrina et al. 2020)) collect per-item submissions but keep them private.

Third, *cultural validity* remains the deeper issue. Most multilingual benchmarks are translations of English-centric constructs. Culturally grounded benchmarks — designed around local educational systems, professional standards, and cultural knowledge — remain rare. KMMLU (Son et al. 2024) (Korean), ArabicMMLU (Koto et al. 2024) (native Arabic questions sourced from regional exams), LawBench (Fei et al. 2023) (Chinese legal system), and TUMLU (Isbarov et al. 2025) (Turkic cultural knowledge) are positive examples; translated MMLU variants are not.

2.6.3 Structural Gaps

Some evaluation challenges are genuinely underserved, not just under-curated.

Long-horizon and multi-session evaluation. Current benchmarks evaluate models on isolated tasks. No benchmark evaluates sustained performance over hours or days, despite this being the primary use case for coding assistants and enterprise agents. The response matrix formulation assumes independent items; long-horizon evaluation introduces temporal dependencies that violate this assumption.

Education as a domain (not just a test source). While many benchmarks use educational exam questions as *items* (MMLU (Hendrycks et al. 2021), C-Eval (Huang et al. 2023), KMMLU (Son et al. 2024)), few evaluate AI *in educational settings*. MathTutorBench (Macina et al. 2025), TutorBench (Srinivasa et al. 2025), and MRBench (Maurya et al. 2025) are emerging efforts, but they are small-scale (under 200 conversations) and focus on pedagogical quality scoring rather than binary correctness. Evaluating tutoring effectiveness, adaptive explanation quality, or long-term learning outcomes requires longitudinal, interactive designs that do not fit the standard response matrix format.

Embodied AI and robotics. Despite substantial work in simulation-based robotics benchmarks (BEHAVIOR-100 (Srivastava et al. 2022), RLBench (James et al. 2020), Meta-World (Yu et al. 2020)), none provide the kind of cross-model response matrices used in this book. The "item" in robotics (a task specification + environment configuration) and the "response" (a trajectory success/failure) could in principle be formalized as a response matrix, but this has not been done at scale.

Audits and compliance evaluation. AI auditing is an increasingly important evaluation modality driven by regulation (EU AI Act, NIST AI RMF, ISO/IEC 42001), but it operates largely outside the response matrix framework. An audit evaluates a *system* against a set of *criteria*, which is structurally a response matrix (systems × criteria → pass/fail), but existing audit data is qualitative and unstandardized. The Responsible AI Measures Dataset (Rismani et al. 2025) catalogs 791 evaluation measures across 11 ethical principles (fairness, transparency, privacy, trust, etc.) extracted from 257 computing papers — essentially an item bank for a future standardized audit instrument. The AI Safety Index (Future of Life Institute 2025) (Future of Life Institute, Winter 2025) provides the closest thing to an audit response matrix: 8 frontier AI

companies scored across 6 safety domains (risk assessment, current harms, safety frameworks, existential safety, governance, information sharing) on a GPA scale. No company scores above a C+; "Existential Safety" is a near-universal F. These are small-scale pilot efforts, but they illustrate the path: if audit criteria were standardized and applied systematically across systems, the psychometric toolkit — IRT for criterion analysis, many-facet Rasch for auditor calibration, factor analysis for dimensional structure — would apply directly.

Post-deployment monitoring. Pre-deployment benchmarks measure what a model *can* do; post-deployment monitoring measures what it *does* do in production. Several structured data sources are emerging. **Incident databases** — the AI Incident Database (McGregor 2021) (AIID, 1,404 incidents with CSET harm taxonomy), the OECD AI Incidents Monitor (OECD 2024) (14,000+ incidents), and the MIT AI Risk Repository (Slattery et al. 2024) (1,700 risk entries from 74 frameworks) — catalog real-world AI failures, though their observational structure (incident logs, not standardized item banks) does not map cleanly to response matrices. **Production interaction logs** offer richer measurement opportunities: WildChat (Zhao et al. 2024) (4.8M real ChatGPT conversations with per-message moderation flags) and LMSYS ToxicChat (Lin et al. 2023) (10K labeled production prompts) effectively provide conversation × safety-category binary matrices from deployed systems. NVIDIA Aegis 2.0 (Ghosh et al. 2025) (33K interactions × 23 hazard categories) provides a similar structure. **Regulatory monitoring data** — notably the NHTSA Standing General Order crash reports (National Highway Traffic Safety Administration 2021) (1,700+ self-driving incidents as downloadable CSVs) and California DMV autonomous vehicle disengagement reports (California Department of Motor Vehicles 2024) (15,000+ disengagements by manufacturer) — provides the most structured post-deployment data, with natural cross-tabulations (manufacturer × incident-type) that function as response matrices. Finally, **model drift studies** — most notably Chen et al. (2024)'s documentation of GPT-4 accuracy dropping from 84% to 51% on certain tasks within three months — reveal that the response matrix itself is not stationary: the same model × item pair can produce different responses over time, violating the IRT assumption that ability parameters are fixed.

Chain-of-thought monitoring and behavioral traces. A rapidly growing body of work monitors AI agents not through their outputs alone but through their internal reasoning processes. OpenAI monitors 99.9% of internal coding agent traffic using GPT-5.4 Thinking to analyze full conversation contexts — user messages, chain-of-thought reasoning, tool calls, and tool outputs — for signs of misalignment, finding that reasoning patterns reveal concerning behaviors (base64 encoding to evade monitors, calling other model versions to bypass restrictions) before they manifest as executed actions. This produces **behavioral traces**: sequential logs of thought → action → observation chains that do not map to response matrices but represent a fundamentally new kind of evaluation data. Several public datasets now capture this structure: Anthropic's alignment faking dataset (Anthropic 2025) (2.14M RL training transcripts with alignment-faking labels, where 34.6% of samples show alignment faking), Scale AI's Monitor Red Teaming dataset (Kale et al. 2025) (MRT, thousands of agent trajectories attempting to evade monitors via lying and manipulation), BELLS (Dorn et al. 2024) (labeled execution traces across hallucination, jailbreak, and prompt injection categories), and the MACHIAVELLI benchmark (Pan et al. 2023) (572K game scenes with dense ethical violation annotations across 25+ categories). CoT faithfulness datasets — FaithCoT-Bench (Shen et al. 2026) (1,000+ annotated trajectories with

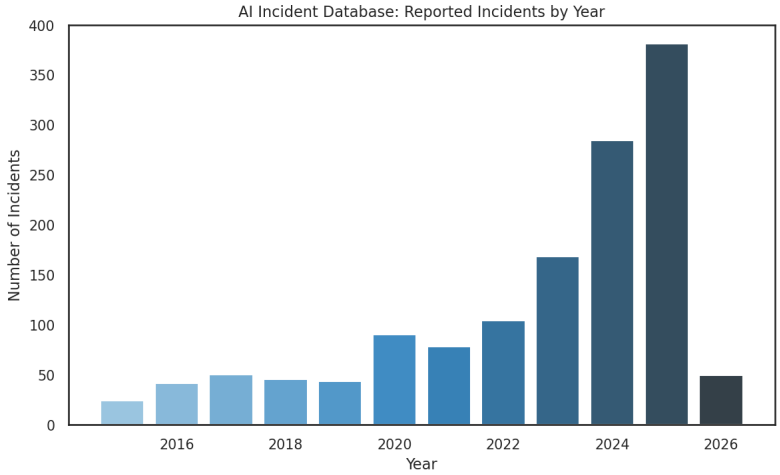


Figure 2.4
Reported AI incidents by year from the AI Incident Database (AIID). The exponential growth — from 24 incidents in 2015 to 381 in 2025 — reflects both increasing deployment of AI systems and increasing reporting infrastructure. The 2026 bar is partial (data through March 2026).

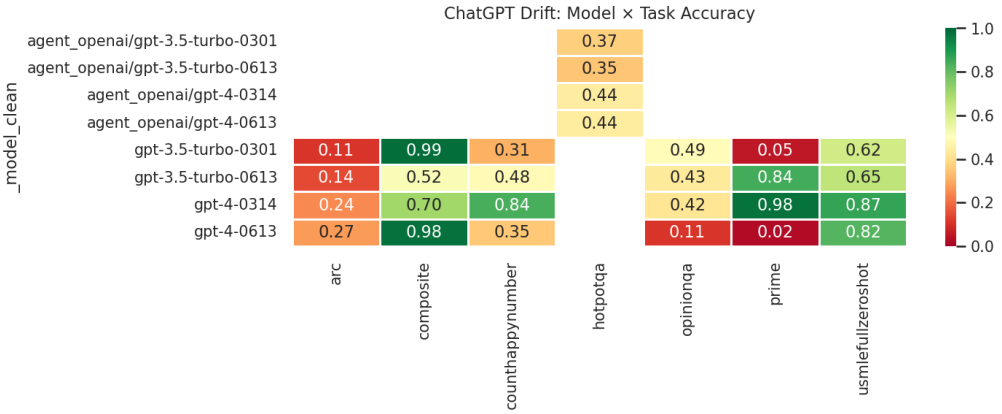


Figure 2.5
ChatGPT drift: model × task accuracy heatmap showing performance degradation across API versions. GPT-3.5-turbo's accuracy on composite tasks dropped from 0.99 (March 2023 version) to 0.14 (June 2023 version). GPT-4 shows similar but less dramatic degradation. This demonstrates that response matrices are not stationary — the same model can produce different responses to the same items over time.

step-level faithfulness labels) and Turpin et al. (2023)'s CoT unfaithfulness experiments — measure whether reasoning chains are faithful to the model's actual decision process. The measurement science challenge: standard IRT assumes independent items, but behavioral traces are inherently sequential and context-dependent. Developing psychometric tools for sequential trace data is an open frontier.

Evaluation of evaluation. Meta-evaluation — assessing whether evaluation methods themselves are reliable and valid — is growing. RewardBench (149 reward models × 2,985 items, now in our collection) (Lambert et al. 2024) evaluates reward models, and the PSN-IRT project (Zhou et al. 2026) constructs a response matrix across 12 models and 41,871 items with full IRT parameter estimation. But systematic studies of whether LLM judges, reward models, and evaluation pipelines are measurement-valid in the sense of Chapter 1 remain rare. Applying the reliability and validity tools from Chapter 5 and Chapter 1 to evaluation methods themselves — not just the systems being evaluated — is an important frontier.

2.7 Data Quality in Practice

Real-world evaluation data is messy. Before fitting measurement models, practitioners must understand and address several recurring data quality issues.

2.7.1 Inconsistent Scoring

Different benchmarks use different scoring conventions, even for similar tasks. Pass@1 may be computed from 1 sample (making it binary), 10 samples (giving 11 discrete values), or 200 samples (approximating a continuous probability). Some benchmarks report accuracy, others report error rate. Some benchmarks score partial credit on multi-step tasks, others use all-or-nothing scoring. Standardizing these into a coherent response matrix requires careful attention to what each score means.

2.7.2 The Subject Identity Problem

For agentic benchmarks, "who is the test-taker?" is not straightforward. Terminal-Bench has 128 agent–model combinations from 31 unique scaffolds and 42 unique models (Merrill et al. 2026). SWE-bench has entries like "SWE-Agent + Claude Sonnet 4" and "OpenHands + Claude Sonnet 4" — same model, different scaffolds, substantially different performance (Jimenez et al. 2023). When we estimate "Claude Sonnet 4's ability," which rows do we use? The answer depends on what construct we are measuring: if we want *model* ability, we should somehow average over scaffolds; if we want *system* ability, each model–scaffold combination is its own subject.

2.7.3 Benchmark Contamination and Temporal Validity

Static benchmarks face an inevitable lifecycle: they are released, adopted by the community, potentially memorized by models trained on web data, and eventually saturated. LiveCodeBench addresses this by drawing problems from ongoing competitions, ensuring items postdate training cutoffs (Jain et al. 2024). But this introduces a new challenge: the item pool changes over time, making longitudinal comparisons difficult without equating procedures.

2.7.4 Missing Data Patterns

Missing data in AI evaluation is rarely random. Common patterns include:

- **Temporal missingness**: older models evaluated on fewer items (LiveCodeBench (Jain et al. 2024))
- **Cost-driven missingness**: expensive models evaluated on fewer benchmarks
- **Availability missingness**: closed-source models not evaluated on benchmarks requiring local execution
- **Selection missingness**: models not evaluated on benchmarks where they are expected to perform poorly

Each pattern violates the missing-at-random assumption that most imputation methods require. The masking schemes developed in Section 7.5 provide a framework for testing how well models handle different missingness patterns.

2.8 The `torch_measure` Toolkit

The `torch_measure` library provides a PyTorch-native toolkit for loading, inspecting, and analyzing response matrices from real benchmarks. It is the companion software for this book.

2.8.1 Loading Benchmarks

```python
from torch_measure.datasets import load, list_datasets

# See all available benchmarks
list_datasets("bench")

# Load a response matrix
rm = load("bench/swebench")
print(f"Shape: {rm.shape}")          # (134, 500)
print(f"Density: {rm.density:.1%}")  # 100.0%
print(f"Mean: {rm.subject_means.mean():.3f}")
```

2.8.2 Inspecting the Response Matrix

```
# Per-subject (model) statistics
print(f"Best model: {rm.subject_means.max():.1%}")
print(f"Worst model: {rm.subject_means.min():.1%}")

# Per-item statistics
print(f"Easiest item: {rm.item_means.max():.1%}")
print(f"Hardest item: {rm.item_means.min():.1%}")
print(f"Items solved by no model: {(rm.item_means == 0).sum()}")

# Missing data
print(f"Missing entries: {(~rm.observed_mask).sum()}")
```

2.8.3 Fitting a Measurement Model

```
from torch_measure.models import Rasch

# Fit a Rasch model to the response matrix
model = Rasch(n_subjects=rm.n_subjects, n_items=rm.n_items)
model.fit(rm.data, method="mle")

# Estimated parameters
abilities = model.ability          # (134,) subject abilities
difficulties = model.difficulty    # (500,) item difficulties

# Predict response probabilities
P_hat = model.predict()            # (134, 500)
```

2.8.4 Comparing Benchmarks

```
# Load multiple benchmarks
benchmarks = {
    "SWE-bench": load("bench/swebench"),
    "BigCodeBench": load("bench/bigcodebench"),
    "BFCL": load("bench/bfcl"),
    "MMLU-Pro": load("bench/mmlupro"),
}

for name, rm in benchmarks.items():
    print(f"{name:15s}: {rm.n_subjects:4d} subjects × {rm.n_items:5d} items, "
          f"density={rm.density:.1%}, mean={rm.subject_means.mean():.3f}")
```

2.8.5 Loading Paired Response Matrices

```
# List intervention datasets
list_datasets("intervention")

# Load paired matrices from Collab-CXR (radiology)
rm_no_ai = load("intervention/collab_cxr_accuracy_no_ai")
rm_with_ai = load("intervention/collab_cxr_accuracy_with_ai")

print(f"No AI:   {rm_no_ai.n_subjects} radiologists × {rm_no_ai.n_items} cases, "
      f"mean accuracy={rm_no_ai.subject_means.mean():.3f}")
print(f"With AI: {rm_with_ai.n_subjects} radiologists × {rm_with_ai.n_items} cases, "
      f"mean accuracy={rm_with_ai.subject_means.mean():.3f}")

# Load GenAI Learning exam data across conditions
for arm in ["control", "augmented", "vanilla"]:
    rm = load(f"intervention/genai_learning_exam_{arm}")
    print(f"Exam ({arm:10s}): {rm.n_subjects} students × {rm.n_items} problems, "
          f"mean={rm.subject_means.mean():.3f}")
```

The response matrices loaded by `torch_measure` are the raw material for everything that follows. Chapter 3 introduces the probabilistic models that decompose these matrices into interpretable latent parameters. Chapter 4 shows how to estimate those parameters. The rest of the book develops the tools for assessing whether the resulting measurements are reliable, valid, and useful.

2.9 Discussion Questions

1. **The subject identity problem.** For SWE-bench, the top-performing entry uses a proprietary scaffold with Claude Opus 4.5 (Jimenez et al. 2023). A competing entry uses an open-source scaffold with the same model and scores 15% lower. Is the difference attributable to the model, the scaffold, or the interaction? How could a study be designed to decompose these contributions? What does this imply for the dimensionality of the response matrix?

2. **Cultural validity of benchmarks.** AfriMedQA evaluates medical knowledge in African healthcare contexts (Nimo et al. 2025). A model trained primarily on US/European medical data might score poorly not because it lacks medical reasoning but because it lacks knowledge of local disease prevalence, treatment protocols, and healthcare infrastructure. Is this a validity threat (construct-irrelevant variance) or a genuine measurement of the construct (medical reasoning *in context*)? How does the answer depend on the intended use?

3. **Dynamic vs. static benchmarks.** LiveCodeBench adds new problems from ongoing competitions to avoid contamination (Jain et al. 2024). But this means a model's "ability"

is estimated from different items at different points in time. Under what conditions is this a problem? How can IRT equating procedures maintain a common scale?

4. **Preference data and transitivity**. The Bradley-Terry model assumes transitive preferences: if $A \succ B$ and $B \succ C$, then $A \succ C$. But human preferences often violate transitivity. How can transitivity violations in Chatbot Arena data be detected and quantified? What are the implications for using Elo ratings as a measure of model quality?

5. **The missing data problem**. In the torch_measure collection, some benchmarks are nearly complete (SWE-bench: 100% density (Jimenez et al. 2023)) while others have systematic gaps (LiveCodeBench: 88.8% (Jain et al. 2024)). How does the missingness pattern affect the validity of IRT ability estimates? Under what conditions are ability estimates from a sparse matrix trustworthy?

6. **Intervention matrices and causal inference**. The GenAI Learning dataset shows that students practicing with ChatGPT score 0.69 during practice but only 0.36 on a no-AI exam — the same as the 0.36 of controls. A naive analysis of the practice phase alone would conclude that AI dramatically improves performance. What does this imply for interpreting human-AI benchmark scores? How could an evaluation framework distinguish AI-augmented performance from durable human capability? Consider the implications for domains beyond education — for instance, would a doctor's diagnostic ability genuinely improve after years of AI-assisted practice, or would it atrophy?

2.10 Bibliographic Notes

The response matrix formulation for AI evaluation draws on the extensive psychometric literature on item analysis and test construction (Lord and Novick 1968; Hambleton and Swaminathan 1985). The systematic curation of response matrices from AI benchmarks is a more recent effort; the `torch_measure` toolkit and its benchmark collection follow the standardized approach described in this chapter.

Individual benchmarks have their own lineages. SWE-bench (Jimenez et al. 2023) established the paradigm of evaluating agents on real GitHub issues. MMLU (Hendrycks et al. 2021) and its successors (MMLU-Pro (Y. Wang et al. 2024)) standardized knowledge evaluation across domains. The Chatbot Arena (Zheng et al. 2023) pioneered crowdsourced pairwise evaluation at scale. LiveCodeBench (Jain et al. 2024) introduced temporal freshness as a design principle for contamination-resistant evaluation.

The paired response matrix framework connects to a broader literature on AI evaluation beyond benchmarks. Deployment RCTs for AI coding assistants include the GitHub Copilot studies (Peng et al. 2023; Cui et al. 2026) and the METR developer productivity study (Becker et al. 2025), which produced the surprising finding that experienced developers were slower with AI assistance. The Collab-CXR dataset (F. Yu et al. 2024) is the largest public reader study with AI-assisted and unassisted conditions. Bastani et al. (Bastani et al. 2025) demonstrate that AI can inflate apparent performance without producing learning. The HAIID study (Vodrahalli et al. 2022) contributes to the literature on algorithmic aversion and appreciation by showing that

advice source labels minimally affect human behavior. For CBRN uplift evaluation, the RAND biosecurity study and OpenAI's early warning system evaluation are the most prominent, though neither releases item-level data. Salaudeen, Koyejo et al. (2025) provide a validity-centered theoretical framework for AI evaluation that motivates the psychometric approach to paired response data taken here.

The multilingual and cultural evaluation efforts respond to a recognized gap in AI measurement. C-Eval (Huang et al. 2023) and CMMLU (H. Li et al. 2023) provide Chinese-language evaluation. KMMLU provides Korean-language evaluation. IberBench covers Iberian languages. AfriEval and AfriMedQA (Nimo et al. 2025) address African languages and healthcare contexts. SIB-200 provides classification across 200+ languages. HELM Multilingual (Bommasani et al. 2024) standardizes evaluation across languages. These efforts collectively demonstrate that the AI evaluation landscape cannot be understood through English-only benchmarks.

For preference data, the Bradley-Terry model dates to Bradley and Milton E. Terry (1952a). The Elo rating system, widely used in the Chatbot Arena, implements online Bradley-Terry estimation. RewardBench (Lambert et al. 2024) standardizes reward model evaluation. The relationship between preference models and measurement theory is developed in the companion text *Machine Learning from Human Preferences* (Truong & Koyejo, 2026).

3 Models

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Distinguish** between Item Response Theory, factor models, paired comparison systems (Elo, Bradley-Terry), and network models (GGM, Ising).
2. **Explain** measurement as a homomorphism (representational measurement theory) and additive conjoint measurement, and map their elements onto the Rasch model.
3. **Explain** why the Rasch model holds a special status as "the measurement model" through the sufficiency of sum scores, specific objectivity, and test-free measurement.
4. **Explain** why observable response probabilities make the conjoint cancellation axioms empirically testable in AI evaluation, and state the caveats (link freedom, multidimensionality, sampling error, and the separation of measurement from validity).
5. **Derive** the sufficiency theorem for the Rasch model and explain its implications for AI benchmark evaluation.
6. **Compare** the prescriptive (Rasch school) and descriptive (general IRT) approaches to measurement and articulate when each is appropriate.
7. **Trace** the historical development from Thurstone (1927) through Rasch (1960) to modern network psychometrics.
8. **Connect** classical measurement concepts (reliability, validity, dimensionality) to contemporary AI benchmark evaluation.
9. **Apply** measurement theory to analyze whether AI benchmarks satisfy the requirements for scientific measurement.
10. **Implement** basic IRT models in Python and visualize item characteristic curves.
11. **Evaluate** the assumptions underlying AI leaderboards and identify potential violations of measurement principles.

Scope of This Chapter

This chapter focuses on **model specification**: we introduce the probabilistic models used for measurement and explain what each one assumes. We do *not* cover how to estimate (learn) the parameters of these models—that is the subject of Chapter 4. Think of this chapter as defining the "what" (the models) and the next chapter as defining the "how" (the estimation).

Video Overview

A visual tour of the key concepts in this chapter — from response matrices and item characteristic curves to factor models and benchmark heterogeneity.

../animations/ch1/chapter1_narrated.mp4

Notation

This chapter introduces the core notation used throughout the book: θ_i (model ability), β_j (item difficulty), $\beta_j = (\beta_{j1}, \beta_{j2}, \beta_{j3})$ (the item-parameter vector collecting difficulty, discrimination, and guessing), Y_{ij} (binary response), and $\sigma(\cdot)$ (logistic sigmoid). **?@sec-notation** collects every symbol in one place—refer to it whenever a symbol is unfamiliar.

Chapter 1 drew the nomological network as a causal diagram: latent constructs θ that produce a system's responses Y, with the loading edges $\theta \to Y$ carrying the meaning we intend to measure. That chapter was conceptual—it said what a valid measurement *requires* (the construct exists and causally produces the responses) but left the edges as bare arrows, with no functional form and no way to recover θ from data. This chapter supplies the form. For the most part of this chapter, we will assume that we have already decided on a construct to be measured and a set of items to target that construct.

The goal is modest: assign each model a single number θ_i that represents its standing on one construct—an *ability*—on a *unidimensional* scale. The hard part is the word *represents*. Computing a model's accuracy already assigns it a number, but a number is not yet a measurement. A bathroom scale reading off body mass and a leaderboard reading off accuracy both produce numbers; only one is trusted to support arithmetic, comparison across instruments, and prediction. This section asks what separates the two—what it takes for "assign a number" to mean *measure*—and arrives at the model that, on one influential account, uniquely delivers it: the Rasch model. We build up in three steps: the general theory of when numbers measure (representational measurement theory, Section 3.1); the specific structure that yields an interval scale for an attribute like ability, and the Rasch model that realizes it (additive conjoint measurement, Section 3.2); and the respect in which AI evaluation can *test* that structure rather than merely assume it (Section 3.5).

3.1 What Does It Mean to Measure?

Representational measurement theory (Krantz et al. 1971) makes precise what it means for numbers to measure. Its central idea: to measure is to construct a structure-preserving map—a *homomorphism*—from an **empirical relational structure** (the objects and the qualitative relations we can actually observe among them) into a **numerical relational structure** (the real numbers with their order and arithmetic).

Why should this be the *central* idea, rather than just "attach a number to each object"? Because the numbers we attach come with a great deal of built-in structure that we cannot help but use. As soon as a model has "ability 1.8" and another "ability 0.9", we are tempted to say the first is *twice* as able, or that the *gap* between them equals some other gap—statements that borrow the order, differences, and ratios of the real line. Those statements are only trustworthy if the

objects genuinely stand in relations that mirror the numerical ones. The empirical relations come *first*: before any numbers, we can observe that this rod balances those two end to end, or that this model beats that one on item after item. A homomorphism is the promise that the numbers are a faithful stand-in for exactly those observations—order maps to order ($\geq$), physical combination maps to addition ($+$)—so that whatever we compute on the numbers can be read back as a true statement about the objects. Measurement, on this view, is not *imposing* a scale but *discovering* a numerical representation that the empirical structure already supports; the numbers are a model of reality the way a map is a model of terrain, trustworthy precisely because distances and directions on the map preserve those on the ground. Assigning numbers that do *not* preserve the structure (jersey numbers on athletes, area codes) is labeling, not measuring: arithmetic on them carries no information back to the world.

Definition: Measurement (representational)

An **empirical relational structure** $\mathcal{E} = \langle A, \succeq, \circ, \dots \rangle$ is a set of objects A with observed relations—an order $\succeq$, perhaps a concatenation operation $\circ$. A **numerical relational structure** is $\mathcal{N} = \langle \mathbb{R}, \geq, +, \dots \rangle$. A **measurement** is a homomorphism $\phi : A \to \mathbb{R}$ preserving those relations:

$$a \succeq b \iff \phi(a) \geq \phi(b), \qquad \phi(a \circ b) = \phi(a) + \phi(b).$$

Two theorems govern any such map.

A **representation theorem** has the form: *if* the empirical structure $\mathcal{E}$ satisfies a list of qualitative axioms, *then* a homomorphism ϕ into the reals exists (and the proof usually constructs one). The axioms are conditions stated entirely in terms of the observable relations—no numbers appear in them—and they are precisely the *empirical shadow* of the properties the number system already has. Real-number order is transitive and complete, so the empirical order $\succeq$ must be too: if $a \succeq b$ and $b \succeq c$ then $a \succeq c$. Real addition is associative and commutative, so if concatenation $\circ$ is to map to $+$, combining objects must be observably associative and commutative—$a \circ b$ must land at the same place on the scale as $b \circ a$.

How rich a scale $\mathcal{E}$ supports depends on how many axioms it satisfies. The recurring ones are:

- **Weak order**. The relation $\succeq$ is *transitive* ($a \succeq b$ and $b \succeq c$ imply $a \succeq c$) and *complete* (every pair is comparable: $a \succeq b$ or $b \succeq a$). Informally: "rank everything, ties allowed."
- **Concatenation** ($\circ$). A way to combine two objects into a third—rods laid end to end, masses heaped on one pan—required to be *associative* ($(a \circ b) \circ c \sim a \circ (b \circ c)$) and *commutative* ($a \circ b \sim b \circ a$), mirroring the same properties of $+$.
- **Monotonicity**. Combining preserves order: $a \succeq b \iff a \circ c \succeq b \circ c$.
- **Archimedean**. No object is infinitely larger than another: enough copies of any object eventually exceed any other.
- **Density** (technical). A countable subset lies order-dense in $\succeq$; needed only to embed *infinite* object sets into $\mathbb{R}$, and automatic for finite benchmarks.

Which scale these buy:

- An **ordinal** scale follows from a weak order (plus density) alone: ϕ preserves order and nothing more, $a \succeq b \iff \phi(a) \geq \phi(b)$.
- A **ratio** scale follows from a weak order *together with* a concatenation operation satisfying associativity, commutativity, monotonicity, and the Archimedean condition; then ϕ also satisfies $\phi(a \circ b) = \phi(a) + \phi(b)$. The construction is essentially counting: fix a unit, lay down copies of it, and record how many balance the object—which is how length and mass are measured.
- An **interval** scale is the awkward middle: it needs additive structure but has *no* concatenation operation to supply it. The next section shows how *two conjoint factors* can play the role concatenation plays here (additive conjoint measurement, Section 3.2).

A **uniqueness theorem** then states how unique the resulting ϕ is—the class of admissible transformations carrying one valid homomorphism to another—and this fixes the **scale type**:

- **Ordinal**: any strictly increasing transformation is admissible; only the *order* of the numbers is meaningful.
- **Interval**: admissible transformations are positive affine, $\phi \mapsto a\phi + b$ with $a > 0$; *differences* (and ratios of differences) are meaningful, but neither the zero nor the unit.
- **Ratio**: admissible transformations are $\phi \mapsto a\phi$; *ratios* are meaningful and there is a true zero.

These types form a hierarchy of increasing structure, each licensing more arithmetic than the last (Stevens 1946). (Below them sits the **nominal** scale—mere labels, like benchmark names or task categories, where only equality is meaningful.) An **ordinal** scale records only rank: medians and percentiles are meaningful, but the spacing between values is not, so a model that ranks first is *better* than the runner-up without being better *by any stated amount*—a raw leaderboard position is ordinal. An **interval** scale adds meaningful distances while keeping an arbitrary zero: differences and averages are licensed but ratios are not, the textbook case being temperature in degrees Celsius (the gap from $10°$ to $20°$ equals the gap from $20°$ to $30°$, yet $20°$ is not "twice as hot" as $10°$); an IRT ability θ on the logit scale is interval in just this sense, its zero a convention. A **ratio** scale adds a true zero, so ratios carry content and "twice as much" means something: length, mass, reaction time, and a raw *count* of items answered correctly are all ratio. The practical warning is that the scale type is fixed by the empirical structure, not by how the numbers look. An accuracy of 0.86 is a perfectly good ratio-scale *count* of correct answers; but read as a measure of *ability* it is at best interval, and without a fitted model only ordinal—the decimals invite ratio and difference comparisons the construct may not support.

This gives "a meaningful numerical value" a precise sense: a numerical statement is **meaningful** exactly when its truth is invariant under the admissible transformations of the scale (Krantz et al. 1971). "Model A scored twice model B" is meaningful on a ratio scale, not an interval one; "the gap from A to B exceeds the gap from C to D" is meaningful on an interval scale, not an ordinal one. Scoring assigns numbers; measurement assigns numbers *and* certifies which statements about them survive rescaling.

Where do our objects of interest fall on this spectrum? Physical *extensive* quantities—length, mass, duration—come with a concatenation operation (lay rods end to end, pile masses on a

balance), and that operation is what earns them a *ratio* scale; physical measurement sits at the rich end. Ability has no such operation: there is no way to concatenate two models', or two people's, abilities into a third. The most an ability scale can aspire to is therefore an *interval* scale, reachable only by the indirect, two-factor route of additive conjoint measurement (Section 3.2). Here human and AI measurement part ways. For human test-takers the conjoint axioms can only be *assumed*, because a response probability is never directly observed—each person meets each item essentially once—so interval status is a modeling commitment, and absent that commitment the defensible reading of a test score is merely *ordinal*. For AI systems the same axioms are *testable*: a model's response probability can be estimated by repeated sampling (Section 3.5), so interval measurement can be earned from data rather than presumed. A leaderboard that only reports who outranked whom sits at the ordinal floor for both.

3.2 Additive Conjoint Measurement

Classical extensive measurement (length, mass) earns its scale from a physical concatenation operation: lay rods end to end and length adds. Latent attributes have no such operation—there is no way to physically concatenate two models' abilities. Additive conjoint measurement (Luce and Tukey 1964) is the branch of representational theory that recovers an interval scale for an attribute jointly determined by two **conjoint factors** from the order of their combined effect. (We call A and X *conjoint factors* to keep them distinct from the unrelated *factor model* of Section 7.2.) Let the conjoint factors be A (models) and X (items), and suppose we can observe, for each pair $(a, x) \in A \times X$, an ordering $\succeq$ by how much of the attribute results (how likely a correct response). Additive conjoint measurement asks when this order on the *product set* admits an additive representation.

Definition: Cancellation axioms

The order $\succeq$ on $A \times X$ satisfies:

- **Single cancellation (independence)**: the order induced on A by holding x fixed does not depend on which x is fixed, and symmetrically for X. So $(a_1, x) \succeq (a_2, x)$ for some x implies $(a_1, x') \succeq (a_2, x')$ for every x'.
- **Double cancellation (the Thomsen condition)**: for all $a_1, a_2, a_3 \in A$ and $x_1, x_2, x_3 \in X$,

$$(a_1, x_2) \succeq (a_2, x_1) \text{ and } (a_2, x_3) \succeq (a_3, x_2) \implies (a_1, x_3) \succeq (a_3, x_1).$$

Single cancellation implies parameter separation. Fixing an item x induces a ranking of models, $a \succeq_x a' :\Leftrightarrow (a, x) \succeq (a', x)$; single cancellation says this ranking is the *same for every* x, so there is one item-independent order on models (and, symmetrically, one model-independent order on items). The conjoint factors decouple —one can speak of "the stronger model" without naming an item. Equivalently, in the additive form the item's contribution is the same constant for both models and cancels: $f(a_1) + g(x) \geq f(a_2) + g(x) \iff f(a_1) \geq f(a_2)$, free of x. A model×item *interaction*—a model strong on mathematics but weak on coding, outranking a rival on some items and trailing it on others—violates single cancellation outright. Double

cancellation then forces the *spacings* of the two conjoint factors to agree, upgrading two separate orderings into one shared additive scale (the content made visible on the 3×3 grid below).

Definition: Solvability and the Archimedean condition

Two further *structural* conditions make the conjoint factors rich enough to carry a real-valued scale:

- **Restricted solvability**: the conjoint factors are finely graded—given a target cell, one can find a level of either factor that matches it, so intervals can always be bisected.
- **Archimedean**: no level is infinitely larger than another—enough equal steps along one factor eventually overtake any difference.

Unlike the cancellation axioms, these are not separately testable on coarse data; they are idealizations that guarantee the construction terminates in $\mathbb{R}$.

Theorem: Additive representation

If $\succeq$ on $A \times X$ satisfies the cancellation, solvability, and Archimedean axioms, then there exist real functions $f : A \to \mathbb{R}$ and $g : X \to \mathbb{R}$ with

$$(a, x) \succeq (a', x') \iff f(a) + g(x) \geq f(a') + g(x').$$

f and g are unique up to a common positive affine transformation ($f \mapsto \alpha f + \beta_1$, $g \mapsto \alpha g + \beta_2$ with the same $\alpha > 0$). They are therefore **interval scales sharing a common unit**.

Proof idea

Single cancellation lets each factor be ordered consistently on its own. Double cancellation is the binding constraint: it forces the *spacings* on A and X to be mutually consistent, so a unit defined on one factor transfers to the other. Restricted solvability supplies intermediate elements to bisect intervals; the Archimedean axiom guarantees the bisection terminates, pinning the scale to the reals. One builds a standard sequence of equally spaced elements and shows the additive representation is forced; common-unit interval uniqueness follows because only the shared unit and the two zeros are free. See Krantz et al. (1971) for the full construction.

The content of double cancellation is easiest to see on a 3×3 grid: three models a_1, a_2, a_3 crossed with three items x_1, x_2, x_3. Each cell (a_i, x_j) has an observed propensity (how likely model i is to get item j right), and *if* the structure is additive that propensity is ordered by $f(a_i) + g(x_j)$. Writing $f_i = f(a_i)$ and $g_j = g(x_j)$, the grid of additive values is:

	x_1	x_2	x_3
a_1	$f_1 + g_1$	$f_1 + g_2$	$f_1 + g_3$
a_2	$f_2 + g_1$	$f_2 + g_2$	$f_2 + g_3$
a_3	$f_3 + g_1$	$f_3 + g_2$	$f_3 + g_3$

Now suppose we observe two comparisons across the off-diagonal cells: $(a_1, x_2) \succeq (a_2, x_1)$

and $(a_2, x_3) \succeq (a_3, x_2)$. Under the additive representation these read $f_1 + g_2 \geq f_2 + g_1$ and $f_2 + g_3 \geq f_3 + g_2$. Add the two inequalities and cancel the common terms f_2 and g_2 from both sides—the two cancellations that give the axiom its name—and what survives is $f_1 + g_3 \geq f_3 + g_1$, or $(a_1, x_3) \succeq (a_3, x_1)$. So additivity forces the third comparison once the first two are seen: the two premises chain $a_1 \to a_2 \to a_3$ across the items, and the conclusion connects the endpoints. If a data set instead shows $(a_3, x_1) \succ (a_1, x_3)$, no functions f, g whatsoever can reproduce it. The violation lives in the ordering of the cells, before any numbers are assigned—which is exactly what makes double cancellation an empirical claim rather than a modeling convenience.

Writing the additive structure inside a monotone link Φ that turns it into a probability gives

$$P(Y_{ij} = 1) = \Phi(\theta_i - \beta_j).$$

Taking Φ to be the logistic $\sigma(z) = 1/(1 + e^{-z})$ yields the Rasch model; taking the standard normal CDF yields the probit model. The additive structure makes the model a *measurement* model of i (and j) – the scale constructed is an interval scale. The monotone link re-expresses the same order as a probability.

In the Rasch model, parameter separation is also discussed as specific objectivity. The additive structure already made model comparisons item-free on the latent scale; the logistic link makes the cancellation in the observable odds,

$$\frac{P(Y_{ij} = 1)/P(Y_{ij} = 0)}{P(Y_{kj} = 1)/P(Y_{kj} = 0)} = \exp(\theta_i - \theta_k),$$

with the difficulty β_j dropping out entirely. Operationally this is test-free comparison and subject-free calibration — a model's ability can be estimated from any subset of calibrated items and an item's difficulty from any sample of models—which the sufficiency of Section 3.3 makes practical (items calibrate without knowing abilities, and conversely). It is what licenses comparing models across different set of items, building reusable item banks, adaptive testing (Chapter 6), and equating benchmarks to a common scale—all contingent, as Section 3.5 stresses, on the Rasch model actually fitting the data.

3.3 Sufficiency and Rasch Model

In AI evaluation we almost always report a model's accuracy collapsing an entire response vector of a subject into a single number. When is that collapse harmless? Only if the detail we throw away— which items were answered correctly, not merely how many—carries no further information about the model's ability. This is the sufficiency assumption, and reporting an average score is only justified if this assumption is valid. Here, we demonstrate that sufficiency assumption further specify the link function ϕ that is unspecified in ACM to be the logistic function $\sigma(\cdot)$. The resulting model is known as the Rasch model:

$$P(Y_{ij} = 1 | \theta_i, \beta_j) = \sigma(\theta_i - \beta_j)$$

Given the latent parameters θ_i and β_j, we assume that responses are conditionally independent. This is also referred to as local independence:

$$P(Y_{i1}, \dots, Y_{iM} \mid \theta_i, \beta) = \prod_{j=1}^{M} P(Y_{ij} \mid \theta_i, \beta_j).$$

Local independence can be tested by residual correlations analysis between item pairs. Local independence can be violated in practice via shared context between items, where multiple items on the same reading passage, dataset split, or few-shot prompt template share information that θ_i does not model.

Below is the plate diagram of the Rasch model:

Reading Plate Diagrams

Throughout this chapter we use **plate diagrams** to visualize probabilistic models, extending the graphical-model conventions of Chapter 1 (open = latent, shaded = observed, arrows = dependence) with one new element—*plates* for replication. The full set of conventions:

- **Shaded (gray) nodes** represent *observed* variables (e.g., responses Y_{ij}).
- **Open (white) nodes** represent *latent* (unobserved) variables or parameters (e.g., ability θ_i).
- **Arrows** indicate probabilistic dependencies: an arrow from A to B means B's distribution depends on A.
- **Plates** (rectangles) indicate replication: a plate labeled "$i = 1, \dots, N$" means the enclosed variables are repeated N times, once per index i.

Nested plates represent crossed or hierarchical structure. For example, the response Y_{ij} sits inside both the person plate (i) and the item plate (j), indicating one observation per person-item pair. For a comprehensive treatment of graphical models, see Koller and Friedman (2009).

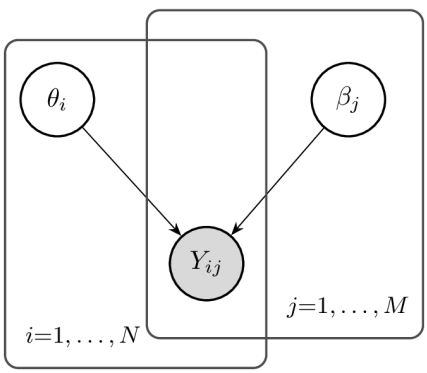


Figure 3.1
Plate diagram for the Rasch model. Shaded nodes are observed; open nodes are latent. Plates indicate replication over persons (i) and items (j).

Next, we show that ACM and sufficiency assumption implies the Rasch model.

Theorem: Additivity + sufficiency imply Rasch

Take an additive latent-trait model in which each response depends on ability and difficulty only through their difference,

$$P(Y_{ij} = 1 \mid \theta_i, \beta_j) = \Phi(\theta_i - \beta_j),$$

for some strictly increasing link Φ—the additive conjoint structure of Section 3.2, which leaves Φ free. If the raw sum score $S_i = \sum_{j=1}^{M} Y_{ij}$ is a sufficient statistic for θ_i, then Φ must be logistic, so the model is the **Rasch model**. The converse also holds, so among additive models sum-score sufficiency *characterizes* Rasch (Andersen 1973; Fischer and Molenaar 1995). Additive structure alone does not deliver sufficiency—the logistic link, singled out here, is what does. (Allowing items to differ in discrimination—the 2PL model—would make a discrimination-weighted score sufficient instead of the raw count.)

Proof

Assume local independence, so the joint probability of a pattern Y_i is $\prod_j \Phi(\theta_i - \beta_j)^{Y_{ij}} \, (1 - \Phi(\theta_i - \beta_j))^{1-Y_{ij}}$, with log-likelihood

$$\sum_j Y_{ij} \, \operatorname{logit} \Phi(\theta_i - \beta_j) \; + \; \sum_j \log(1 - \Phi(\theta_i - \beta_j)), \qquad \operatorname{logit}(p) = \log \tfrac{p}{1-p}.$$

The second sum depends on θ_i but not on *which* items were correct, so it enters only the normalizer. By the factorization theorem, $S_i = \sum_j Y_{ij}$ is sufficient for θ_i exactly when θ_i enters the first sum *only through* S_i—that is, when $\operatorname{logit} \Phi(\theta_i - \beta_j)$ splits as $c\,\theta_i + d_j$ with the *same* slope c for every item. A function of the single argument $\theta_i - \beta_j$ has that form only if $\operatorname{logit} \Phi$ is linear, i.e. $\Phi(z) = 1/(1 + e^{-cz})$; absorbing c into the scale gives the logistic link—the Rasch model.

Conversely, in the Rasch model

$$P(Y_i \mid \theta_i, \beta) = \frac{\exp(\theta_i S_i - \sum_j Y_{ij} \beta_j)}{\prod_j (1 + \exp(\theta_i - \beta_j))}$$

factors as $g(S_i, \theta_i)\, h(Y_i, \beta)$, so S_i is sufficient; and the $\exp(\theta_i S_i)$ factor cancels in $P(Y_i \mid S_i, \theta_i) = P(Y_i \mid \theta_i)/P(S_i \mid \theta_i)$, leaving the conditional pattern distribution free of θ.

The implication is that, before trusting average benchmark scores, we should test whether the Rasch model fits the data.

3.4 Reinterpreting the Second Conjoint Factor

In some evaluation settings, we observe *pairwise comparisons*: which of two items is preferred, which of two players wins. The Bradley–Terry model (1952) is the foundational model for paired comparisons. As we will see, it is not a new model at all but the Rasch structure of Section 3.2 with its **second conjoint factor reinterpreted**—the second slot holds not an item (carrying a difficulty) but a competitor (carrying a strength), drawn from the same pool as the first:

$$P(i \succ j) = \sigma(\theta_i - \theta_j)$$

where $i \succ j$ reads "i is preferred to (beats) j" and θ_i, θ_j are the "strength" or "quality" of competitors i and j, both drawn from the same pool. Mathematically this *is* the Rasch model; only the reading of the second conjoint factor changes—instead of a subject answering an item, we have two subjects competing against each other.

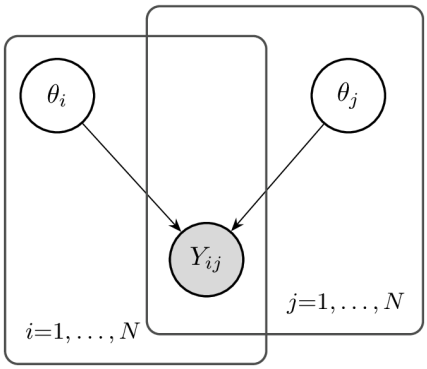


Figure 3.2
Plate diagram for the Bradley-Terry model, sharing the crossed-plate structure of the Rasch model (Figure 3.1): the comparison outcome Y_{ij} sits in the overlap of two competitor plates and depends on the strengths θ_i and θ_j. Bradley-Terry is exactly Rasch with the item difficulty β_j replaced by a second competitor strength θ_j, both competitors drawn from the same pool of N.

The shared form with Rasch is not a coincidence of notation—it means the Bradley-Terry strength inherits the *interval* scale that additive conjoint measurement earns for Rasch ability (Section 3.2). Because the win probability depends only on the *difference* $\theta_i - \theta_j$ passed through a monotone link, it carries exactly the additive conjoint structure of Section 3.2, now with the two conjoint factors being the two *roles* of a comparison—the competitor in the first slot and the competitor in the second—so that the log-odds $\theta_i - \theta_j$ splits additively into a first-role contribution $f(i) = \theta_i$ and a second-role contribution $g(j) = -\theta_j$. Single cancellation holds: the ranking of first-slot competitors induced by any fixed opponent is the same opponent-independent order, because it depends on θ_i alone. Double cancellation holds because the structure is additive by construction. By the representation theorem (Section 3.2; Krantz et al. (1971)), the strengths θ therefore form an interval scale. Because the two roles range over the *same* set of competitors, this is the special "difference measurement" case of conjoint measurement (Krantz et al. 1971), in which a single scale θ plays both conjoint factors at once.

The interval reading has teeth. *Differences* are meaningful—$\theta_i - \theta_j$ is precisely what fixes the win probability $\sigma(\theta_i - \theta_j)$—but there is no true zero, so *ratios* are not: a fixed rating gap means the same win probability wherever it sits on the scale, yet a competitor rated 1600 is not "twice as strong" as one rated 800. Only the origin is conventional (the logistic link pins the unit, exactly as in Rasch; Elo fixes the origin near 1500). This is the paired-comparison counterpart of Rasch's specific objectivity: just as a model's Rasch ability can be estimated from any subset of items, a competitor's strength can be estimated from any schedule of opponents. And as with

Rasch, AI evaluation can *test* this structure rather than assume it: repeated comparisons estimate each $P(i \succ j)$ directly, so additivity (double cancellation) can be checked on the array of win rates by the argument of Section 3.5. When the check fails the diagnosis parallels the Rasch case: inconsistent *spacings* cost only the interval scale while leaving a consistent ranking, but *non-transitive* matchups—A beats B, B beats C, yet C beats A, the paired-comparison signature of multidimensional or "rock–paper–scissors" capabilities—break even the ranking, signalling that no single strength number can order the field.

The Elo rating system, developed by Arpad Elo for chess ratings, is a Bradley-Terry model with online updates. After player i with rating R_i plays player j with rating R_j, the ratings are updated: $R_i^{\text{new}} = R_i + K(y_i - E_i)$, where $Y_i \in \{0, 0.5, 1\}$ is the outcome (loss, draw, win), $E_i = \sigma((R_j - R_i)/400 \cdot \ln 10)$ is the expected outcome, and K is a learning rate parameter. The Elo system is widely used in competitive games and has been adopted for AI evaluation in settings like the Chatbot Arena, where humans compare model outputs pairwise. The Chatbot Arena (LMSYS) uses Elo ratings to rank language models based on human preferences. When a user prefers model A's response over model B's, this is treated as a "win" for model A. The resulting ratings provide a preference-based complement to accuracy-based benchmarks.

3.5 Testing for an Interval Scale

In human testing, the response probability $p_{ij} = P(Y_{ij} = 1)$ is unobservable. A person cannot be given the same item repeatedly under independent conditions—memory, learning, and fatigue destroy the independence—so only a single binary Y_{ij} is ever seen, and p_{ij} must be *inferred* through a model rather than measured. This is why the conjoint axioms of Section 3.2 could only be assumed for human testing: the deterministic order on $A \times X$ that the axioms constrain was never directly observable. AI evaluation breaks this premise. A stateless model queried at sampling temperature > 0 can be given the same item T times with genuinely independent responses, so $\hat{p}_{ij} = \frac{1}{T}\sum_t Y_{ijt}$ estimates p_{ij} *itself*, and T can be made large. The observable object is then not a single bit but a real-valued array $[\hat{p}_{ij}]$ over the model×item grid—exactly the conjoint structure the cancellation axioms describe. The deterministic-versus-probabilistic gap of Section 3.2 narrows to a testable question:

$$\text{additive conjoint structure holds} \iff \exists\ \text{monotone } \Phi : \ \Phi^{-1}(p_{ij}) \text{ is additive } (\theta_i + \delta_j),$$

equivalently, the array $[p_{ij}]$ satisfies double cancellation. The Rasch model is the special case in which the additive scale is the logit. So in AI evaluation we can, for the first time, *test* whether ability and difficulty realize additive conjoint measurement instead of assuming it—by checking double cancellation directly, or by finding the monotone transform that minimizes departure from additivity and asking whether the residual interaction is within sampling error.

That phrase—"find the monotone transform that minimizes departure from additivity"—is, concretely, an **isotonic regression**, and naming it that way makes the whole test transparent. The Rasch model bundles two assumptions: the additive index $\theta_i - \beta_j$ and the logistic *shape* σ. A bad Rasch fit cannot tell you which one broke. To test additivity *alone*, keep the index and let the shape go free.

Definition: Nonparametric additive model (ADISOP)

$$P(Y_{ij} = 1 \mid \theta_i, \beta_j) = F(\theta_i - \beta_j),$$

where θ_i (ability) and β_j (difficulty) lie on a common scale and F is a *single, unknown, non-decreasing* function—the link—estimated from the data rather than assumed. Every item is the same curve F shifted by β_j, so the item characteristic curves are **parallel**; only their shared shape is free. Equivalently, this is a monotone *single-index* model: the entire model × item grid is summarized by one scalar index $s_{ij} = \theta_i - \beta_j$, and the response probability depends on the grid *only* through s_{ij}, monotonically. It is Scheiblechner's ADISOP (Scheiblechner 1999); the order-restricted conjoint tests of Karabatsos (2001) ask the same question.

Estimating F *is* an isotonic regression. Hold the parameters fixed, place every cell's observed $\hat{p}_{ij}$ on the axis given by its index $s_{ij} = \theta_i - \beta_j$, and fit the non-decreasing step function that best matches them—the pool-adjacent-violators solution. Then update θ, β to sharpen the fit and iterate. Because the *only* freedom beyond the additive index is the monotone shape, this can reproduce $[\hat{p}_{ij}]$ **if and only if** the response probability really does depend on the grid through $\theta_i - \beta_j$ alone—which is double cancellation. What the isotonic fit *cannot* absorb is any dependence on the grid *beyond* the one-dimensional index: a genuine model × item interaction. So the **residual of the isotonic regression is the additivity test**, and it forks the Rasch ambiguity cleanly:

- **Small residual, but Rasch fit badly**. The data are additive; Rasch's misfit was the logistic *shape*, which the isotonic F has now replaced. Additive conjoint measurement holds—the interval scale stands—you have merely *estimated* the link instead of assuming it, forgoing the sum-score sufficiency that was the logistic's private bonus (Section 3.3).
- **Large residual even after the best isotonic** F. No monotone link makes the data additive; the dependence on the grid is genuinely two-dimensional, and additive conjoint measurement is rejected (what that costs, and how to localize it, is the next question).

One discipline keeps this honest. The isotonic F is flexible—it will fit *at least* as well as Rasch—so a smaller residual alone proves nothing. The sharp comparison is against the model that breaks additivity with the *fewest* extra knobs: the 2PL, which adds one slope a_j per item. On the logit scale the 2PL index is $a_j\theta_i - b_j$, whose slope varies by item; that item-specific stretching is exactly the dependence a single index $\theta_i - \beta_j$ cannot carry, and it makes the parallel curves cross. So if the parallel single-index fit already suffices, additivity holds; if you genuinely need per-item slopes, the unit of θ is item-dependent and additivity fails.

The model sits on a ladder of decreasing commitment, each rung an isotonic constraint loosened: Rasch (additive index + logistic link) ⊂ the nonparametric additive model (additive index + isotonic link, still interval) ⊂ the doubly-monotone model (non-crossing curves, ordinal specific objectivity) ⊂ the monotone model. Each step right gives up an assumption and weakens the scale, interval to ordinal. In practice these isotonic models are measurement-theory tools with limited software; probit is the cheap parametric stand-in that keeps the additive index while only swapping the link.

What does *failing* mean? Only that no monotone Φ makes $[\hat{p}_{ij}]$ additive: the items admit no common interval scale on which a single ability orders every model. A failed test withdraws exactly one claim—that this item set measures *one* latent quantity, on these models, at the interval level—and it is a verdict on the *grid* tested (these items crossed with these models), not on the instrument in the abstract: a different population of models, or a unidimensional sub-domain of items, may still pass. Nor does it cost everything. If models rank the same way on every item but the *spacings* fail to align, one keeps a consistent *ordinal* ranking—accuracy still orders models, in the stochastic-ordering sense—and forfeits only the interval scale; the ranking itself breaks down only when models *reorder* across items.

These two cases are distinguished by their cause:

1. **Bad items**. Miskeyed, contaminated, or off-construct items create cancellation violations; removing them may restore additivity. This is the curation the diagnostics of Section 3.3 perform.
2. **Genuine multidimensionality**. A model strong at mathematics but weak at coding produces systematic interactions that *no* monotone transform removes. Here the failure is substantive: ability is not one quantity on this item set, and the honest response is to restrict the claim to a unidimensional sub-domain or adopt a multidimensional model (Section 7.2).

Three caveats keep the test honest. First, the link is free: passing double cancellation gives additive conjoint measurement, of which Rasch (logit) and probit are two coordinate systems—not Rasch specifically. Second, one observes $\hat{p}_{ij}$ with $O(1/\sqrt{T})$ error, so the deterministic axioms must be tested in their order-restricted, probabilistic form (Karabatsos 2001; Scheiblechner 1995); the remedy—raise T—is available in AI but not in human testing. Third, one can almost always *force* additivity by deleting enough items, even from noise, so a "pass" counts only if the curation was principled and additivity survives beyond chance.

Finally, passing the test certifies *quantitative structure*, not *validity*. An array can be perfectly additive in an attribute that is not the construct we intended—the scraped-coding benchmark of Chapter 1 has additive structure in *memorization*, not coding ability. Additive conjoint measurement answers "is this a quantity, and may I do interval arithmetic on it?"; it does not answer "is it the quantity I claim?" That second license is construct validity, and it must be earned separately.

The additive structure leaves a visible fingerprint. Take a real response matrix Y (Section 2.1): 50 models from the Open LLM Leaderboard, each scored on the same 100-item sample, with entry 1 for a correct answer and 0 otherwise. The raw matrix looks unstructured, but sorting its rows and columns by their totals exposes the conjoint order.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
```

```
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: response-matrix-visualization
#| autorun: true
#| fig-cap: "A real response matrix: 50 models from the Open LLM Leaderboard scored on
 ↪  a 100-item sample. Rows are sorted by model accuracy, columns by item
 ↪  proportion-correct. The near-triangular gradient is the empirical signature of an
 ↪  additive ability--difficulty structure ordering both models and items."

import json
from pyodide.http import open_url

# Real 50-model x 100-item slice of the Open LLM Leaderboard (binary scored).
# Loaded from a precomputed resource shipped with the book.
raw = json.loads(open_url("data/openllm_subset.json").read())
Y = np.array(raw["data"], dtype=float)

# Sort models (rows) and items (columns) by their totals
row_order = np.argsort(Y.sum(axis=1))[::-1]
col_order = np.argsort(Y.sum(axis=0))[::-1]
Y_sorted = Y[row_order][:, col_order]

fig, ax = plt.subplots(1, 1)
ax.imshow(Y_sorted, aspect='auto', cmap='Blues', interpolation='nearest')
ax.set_xlabel('Items (sorted by proportion correct)')
ax.set_ylabel('Models (sorted by accuracy)')
ax.set_title('Open LLM Leaderboard: sorted response matrix (50 × 100)')

plt.show()
```

In the sorted matrix the correct answers concentrate toward one corner and thin out toward the opposite—a near-triangular gradient rather than a uniform scatter. This is the conjoint order made visible: models range from those that succeed on nearly every item to those that succeed only on the easiest, items range from those nearly every model solves to those almost none do, and a single additive $\theta_i - \beta_j$ orders the whole grid. The gradient need not appear—multidimensional or noisy data wash it out—which is precisely the point: additivity is a property

the data can fail to have.

A caution about the word "realization." The Rasch model is the *probabilistic analogue* of additive conjoint measurement (Brogden 1977; Perline et al. 1979), not a deduction from its axioms. The cancellation axioms describe a deterministic order on $A \times X$; the Rasch model specifies probabilities. Bridging the two—reading a deterministic conjoint structure off stochastic responses—is exactly the contested step (Michell 2008; Kyngdon 2008). The additivity lives in the model's parameters, not, without further work, demonstrably in the world. The next two sections close this gap from both sides: the statistical characterization that makes the structure *internal* to the model (Section 3.3), and the feature of AI evaluation that makes the conjoint axioms *empirically testable* (Section 3.5).

Application: Using Sufficiency to Find Benchmark Bugs

S. T. Truong et al. (2025) turn the sufficiency property into a practical diagnostic tool. Their argument: if the AI evaluation community reports mean scores as the primary metric, it is *implicitly assuming* that the sum score is a sufficient statistic for ability — which, by the theorem above, implies the Rasch model is the data-generating process. Under the Rasch model, two testable consequences follow:

1. **Positive tetrachoric correlations**. All inter-item tetrachoric correlations must be non-negative (Corollary of Chebyshev's inequality applied to increasing functions of the same latent variable θ).
2. **Positive item-total correlations**. Each item must correlate positively with the total score.

Items that violate these conditions — negative tetrachoric correlations, negative item-total correlations, or low Mokken scalability coefficients — are flagged as potentially invalid. Applying this to nine benchmarks including GSM8K and MMLU, S. T. Truong et al. (2025) achieve up to 84% precision at the top-50 flagged items: of the 50 most suspicious questions, up to 42 were confirmed invalid by human experts. Common errors include ambiguous wording, incorrect answer keys (e.g., treating exponential depreciation as linear), grading bugs (e.g., "\$7.00" $\neq$ "7"), and culturally-embedded assumptions.

This illustrates a broader principle: the mathematical properties of measurement models are not merely theoretical — they yield *operational* diagnostics for evaluation quality. When data violate model predictions, either the model is wrong or the data are corrupted. Since the sufficiency assumption is already implicit in how the community uses benchmarks, violations are strong evidence of item-level problems.

To see the diagnostic in action, plant eight defective items in a simulated Rasch benchmark—four with flipped answer keys, four answered at random—and rank every item by its **item–total correlation** (its correlation with the rest of the test). Sound items show the *positive manifold*; the defects fall to the bottom.

```
#| label: item-bug-diagnostic
#| autorun: true
#| fig-cap: "Item--total correlation for every item in a simulated benchmark with
 ↪  eight planted defects (red): flipped-key items correlate negatively, random items
 ↪  sit near zero, and sound items (blue) sit well above. Ranking by this correlation
 ↪  flags the defects---the @truong2025bugs diagnostic in miniature."
import numpy as np
```

```
import matplotlib.pyplot as plt

def sigmoid(x):
    return np.where(x >= 0, 1 / (1 + np.exp(-x)), np.exp(x) / (1 + np.exp(x)))

rng = np.random.RandomState(0)
N, M, n_bad = 200, 60, 8
theta = rng.normal(0, 1, N)
beta  = rng.normal(0, 1, M)
Y = (rng.rand(N, M) < sigmoid(theta[:, None] - beta[None, :])).astype(float)

bad = rng.choice(M, n_bad, replace=False)
for idx, j in enumerate(bad):
    if idx < n_bad // 2:
        Y[:, j] = 1 - Y[:, j]                       # flipped answer key
    else:
        Y[:, j] = (rng.rand(N) < 0.5).astype(float)  # random responses (no signal)

total = Y.sum(1)
itc = np.array([np.corrcoef(Y[:, j], total - Y[:, j])[0, 1] for j in range(M)])  #
 ↪  leave-one-out
is_bad = np.zeros(M, bool); is_bad[bad] = True

order   = np.argsort(itc)
flagged = order[:n_bad]
precision = np.mean(is_bad[flagged])

plt.figure(figsize=(3.6, 2.2))
plt.scatter(range(M), itc[order],
            c=['#d62728' if is_bad[i] else '#1f77b4' for i in order], s=12)
plt.axhline(0, color='black', lw=0.8)
plt.xlabel('item (ranked by item--total correlation)')
plt.ylabel('item--total correlation')
plt.title('Broken items have low correlation'); plt.grid(True, alpha=0.3)
plt.tight_layout(); plt.show()
print(f"precision@{n_bad} (flag the lowest-correlation items) = {precision:.2f}")
```

The defective items fall to the bottom, and flagging the lowest-correlation items recovers all eight. In practice the flagged items should be *inspected* rather than deleted blindly, since a low correlation can also mark a genuinely multidimensional skill rather than an error.

3.6 The n-Parameter Logistic Model

Sometimes, the data does not fit the Rasch model, even for the isotonic version. In that case, we cannot license an interval scale, but we can still construct a useful ordinal scale with a more statistically expressive class of models. The n-parameter logistic model is such a class of model. The n-parameter logistic model equips each item with a parameter vector $\beta_j = [\beta_{j1}, \dots, \beta_{jn}] \in$

$\mathbb{R}^n$, letting items differ along more than one axis. Rasch is the case $n = 1$; the cases $n = 2$ and $n = 3$ below add a discrimination and a guessing component, respectively. Because every variant still attaches a single parameter object to each item, the plate diagram is structurally identical to the Rasch model (Figure 3.1)—the lone item node β_j now standing for the full vector β_{j} rather than a scalar difficulty.

The two-parameter logistic model gives item j a discrimination β_{j2} alongside its difficulty β_{j1}:

$$P(Y_{ij} = 1|\theta_i, \beta_{j}) = \sigma(\beta_{j2}(\theta_i - \beta_{j1}))$$

Items with higher discrimination are better at distinguishing between persons of different abilities. Their ICCs are steeper, meaning small changes in ability produce large changes in response probability.

The per-item discrimination has a precise consequence for the *scale type* of the ability estimate. In the 2PL model, logit $P(Y_{ij} = 1) = \beta_{j2}(\theta_i - \beta_{j1})$, so the log-odds gap between two models on item j is $\beta_{j2}(\theta_i - \theta_k)$. Its *sign* is the same on every item (discriminations are positive), so models keep a consistent ordering—single cancellation, the ordinal separation of Section 3.2, survives. But its *magnitude* is scaled by the item's discrimination β_{j2}: each item converts θ-differences into log-odds at its own rate, so the *unit* of θ is item-dependent. Double cancellation fails, no additive representation exists, and θ is therefore **not a fundamental (additive conjoint) interval scale**. It is identified only up to a positive affine transformation—one fixes its origin and unit by convention, typically $\theta \sim \mathcal{N}(0, 1)$—so it is *interval by parametric stipulation*, not because the data carry an item-independent unit. Rasch is exactly the case $\beta_{j2} \equiv 1$: one common exchange rate, an item-independent unit, a genuine interval scale. This sharpens both camps. The prescriptive claim becomes precise—only Rasch earns the interval scale from structure, while the 2PL borrows it from the assumed logistic–Gaussian form. So does the descriptive reply: an affine-identified θ that fits the data is interval enough for ranking and selection, and discarding items to force $\beta_{j2} \equiv 1$ throws away real information about which items discriminate.

For multiple-choice tests, even low-ability test-takers may answer correctly by guessing. The **three-parameter logistic model** adds a lower asymptote, a guessing parameter β_{j3}, so $\beta_{j} = (\beta_{j1}, \beta_{j2}, \beta_{j3})$:

$$P(Y_{ij} = 1|\theta_i, \beta_{j}) = \beta_{j3} + (1 - \beta_{j3})\sigma(\beta_{j2}(\theta_i - \beta_{j1}))$$

where $\beta_{j3} \in [0, 1]$ is the guessing (or pseudo-chance) parameter. For a 4-option multiple-choice item, we might expect $\beta_{j3} \approx 0.25$ if low-ability test-takers guess randomly.

```
#| label: icc-models
#| autorun: true
```

```
#| fig-cap: "Item characteristic curves (ICCs) across the logistic IRT models---each curve gives the probability of a correct response as a function of ability for a fixed item. Left: Rasch (1PL) at several difficulties, the same curve shifted in location. Middle: 2PL at a common difficulty but different discriminations, where a steeper slope distinguishes ability levels more sharply. Right: 1PL, 2PL, and 3PL compared, the 3PL adding a non-zero lower asymptote for guessing."

import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

theta = np.linspace(-4, 4, 200)

fig, axes = plt.subplots(1, 3, figsize=(6, 2), sharey=True)
legend_kw = dict(loc='upper left', fontsize=6, framealpha=0.9,
                 borderpad=0.2, labelspacing=0.2, handlelength=1.2, handletextpad=0.4)

# Left panel: Rasch (1PL), varying difficulty
for beta in [-2, -1, 0, 1, 2]:
    axes[0].plot(theta, sigmoid(theta - beta), label=f'$\\beta = {beta}$', linewidth=1.5)
axes[0].axhline(y=0.5, color='gray', linestyle='--', alpha=0.5)
axes[0].set_xlabel('Ability ($\\theta$)', fontsize=8)
axes[0].set_ylabel('$P(Y = 1)$', fontsize=8)
axes[0].set_title('1PL: effect of difficulty', fontsize=9)
axes[0].legend(title='Difficulty', title_fontsize=6, **legend_kw)
axes[0].tick_params(labelsize=7)
axes[0].grid(True, alpha=0.3)

# Middle panel: 2PL, common difficulty, varying discrimination (y-axis shared)
beta = 0
for a in [0.5, 1.0, 1.5, 2.0]:
    axes[1].plot(theta, sigmoid(a * (theta - beta)), label=f'$a = {a}$', linewidth=1.5)
axes[1].axhline(y=0.5, color='gray', linestyle='--', alpha=0.5)
axes[1].axvline(x=0, color='gray', linestyle='--', alpha=0.5)
axes[1].set_xlabel('Ability ($\\theta$)', fontsize=8)
axes[1].set_title('2PL: effect of discrimination', fontsize=9)
axes[1].legend(title='Discrimination', title_fontsize=6, **legend_kw)
axes[1].tick_params(labelsize=7)
axes[1].grid(True, alpha=0.3)

# Right panel: 1PL vs 2PL vs 3PL at a fixed difficulty (y-axis shared)
a, beta, c = 1.5, 0, 0.25
axes[2].plot(theta, sigmoid(theta - beta), label='1PL (Rasch)', linewidth=1.5)
axes[2].plot(theta, sigmoid(a * (theta - beta)), label='2PL', linewidth=1.5)
```

```
axes[2].plot(theta, c + (1 - c) * sigmoid(a * (theta - beta)), label='3PL',
 ↪  linewidth=1.5)
axes[2].axhline(y=0.5, color='gray', linestyle='--', alpha=0.3)
axes[2].axhline(y=c, color='gray', linestyle=':', alpha=0.5, label=f'Guessing = {c}')
axes[2].set_xlabel('Ability ($\\theta$)', fontsize=8)
axes[2].set_title('1PL vs 2PL vs 3PL', fontsize=9)
axes[2].legend(**legend_kw)
axes[2].tick_params(labelsize=7)
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

3.7 Exercises

Exercise 1.1 (*): Explain why the sum score is a sufficient statistic in the Rasch model but not in the 2PL model. What information is lost when we reduce responses to sum scores under 2PL?

Exercise 1.2 (**): Prove that the Bradley-Terry model is equivalent to a Rasch model where each "person" is a comparison between two items.

Hint: Consider a "person" as an ordered pair (i, j) representing a comparison, and an "item" as a single entity k appearing in a comparison. Define appropriate ability and difficulty parameters.

Exercise 1.3 (**): Show that in the Rasch model, the odds ratio for persons i and k responding correctly to item j is:

$$\frac{P(Y_{ij} = 1)/P(Y_{ij} = 0)}{P(Y_{kj} = 1)/P(Y_{kj} = 0)} = \exp(\theta_i - \theta_k)$$

independent of the item difficulty β_j. Explain why this property is called "specific objectivity."

Exercise 1.4 (***): The Ising model and the Rasch model make different assumptions about why responses correlate.

(a) Write down both models for binary data $Y \in \{0, 1\}^{N \times M}$.

(b) Describe the causal structure each model assumes.

(c) Under what conditions might each model be appropriate for AI evaluation?

(d) Propose an empirical test that could distinguish between them.

Exercise 1.5 (**): Implement Rasch model estimation using conditional maximum likelihood.

```
# Given: Response matrix Y (N models x M questions)
# Task: Estimate item difficulties using conditional MLE
#
# Steps:
# 1. Compute sum scores for each model
# 2. For each item, compute the conditional likelihood given sum scores
# 3. Optimize to find item difficulties
# 4. Compare estimated difficulties to empirical item means (proportion correct)
#
# Use scipy.optimize.minimize for optimization

import numpy as np
from scipy.optimize import minimize
from scipy.special import logsumexp

def estimate_rasch_conditional(Y):
    """
    Estimate Rasch model item difficulties using conditional MLE.

    Parameters:
    -----------
    Y : np.ndarray, shape (N, M)
        Binary response matrix

    Returns:
    --------
    beta : np.ndarray, shape (M,)
        Estimated item difficulties (identified by setting sum(beta) = 0)
    """
    N, M = Y.shape
    # CODE HERE
    pass

# Test on simulated data
np.random.seed(42)
N, M = 100, 50
theta_true = np.random.normal(0, 1, N)
beta_true = np.random.normal(0, 1, M)
prob = 1 / (1 + np.exp(-(theta_true[:, None] - beta_true[None, :])))
Y = (np.random.random((N, M)) < prob).astype(int)

beta_hat = estimate_rasch_conditional(Y)
# Compare to true values (after centering)
```

Exercise 1.6 (**): Given pairwise preference data, estimate Bradley-Terry parameters.

```
# Given: Comparison data as list of (winner, loser) pairs
# Task: Estimate strength parameters via maximum likelihood
```

```
#
# The likelihood for comparison (i beats j) is:
# P(i > j) = exp(theta_i) / (exp(theta_i) + exp(theta_j))
#          = sigmoid(theta_i - theta_j)

import numpy as np
from scipy.optimize import minimize

def estimate_bradley_terry(comparisons, n_items):
    """
    Estimate Bradley-Terry model parameters.

    Parameters:
    -----------
    comparisons : list of (int, int)
        List of (winner, loser) pairs
    n_items : int
        Number of items

    Returns:
    --------
    theta : np.ndarray, shape (n_items,)
        Estimated strength parameters (identified by setting theta[0] = 0)
    """
    # CODE HERE
    pass

# Test: Simulate comparisons and recover parameters
```

Exercise 1.7 (***): Test whether benchmark data fit the Rasch model using Andersen's likelihood ratio test.

```
# Andersen's LR test:
# 1. Split persons into groups based on sum score (e.g., high vs low scorers)
# 2. Estimate item difficulties separately for each group
# 3. If Rasch holds, these estimates should be equal
# 4. Test statistic: 2 * (sum of group log-likelihoods - pooled log-likelihood)
# 5. Under H0, this is chi-squared with df = (n_groups - 1) * (n_items - 1)

def andersen_lr_test(Y, n_groups=2):
    """
    Perform Andersen's LR test for Rasch model fit.

    Parameters:
    -----------
    Y : np.ndarray, shape (N, M)
        Binary response matrix
    n_groups : int
```

```
        Number of groups to split persons into

    Returns:
    --------
    statistic : float
        LR test statistic
    p_value : float
        p-value from chi-squared distribution
    """
    # CODE HERE
    pass
```

Exercise 1.8 (***): Implement the benchmark item-bug diagnostic of S. T. Truong et al. (2025).

```
# Simulate a Rasch benchmark with planted defective items, then rank items by
# three scalability diagnostics and measure how well each recovers the defects.
#
# Steps:
# 1. Simulate Y (N = 80 models x M = 200 items) from a Rasch model.
# 2. Plant 20 buggy items: 10 with flipped keys, 10 answered at random.
# 3. Rank items by (a) item-total correlation, (b) average tetrachoric
#    correlation, and (c) Mokken H_j (@eq-mokken-h).
# 4. Report precision@20 and recall@20 for each ranking.
# 5. Repeat as the model pool grows from N = 10 to N = 100 and plot how
#    detection quality scales with the number of test-takers.

import numpy as np

def detect_buggy_items(N, M, n_bad, seed=0):
    """Return precision@n_bad and recall@n_bad for each of the three diagnostics."""
    # CODE HERE
    pass
```

Discussion 1.1: The Rasch model's sufficiency property justifies using sum scores as measurements. But most AI benchmarks use items with varying discriminations, which violates the Rasch assumption. Does this mean that current benchmark rankings are fundamentally flawed, or can they still be useful approximations? Under what conditions would adopting Rasch-based measurement change the ranking of frontier models?

Discussion 1.2: The Rasch school argues that items not fitting the Rasch model should be discarded because they do not measure the same construct. What are the implications of this view for AI benchmark design? Should we design benchmarks to fit Rasch, or should we use more flexible models that accommodate heterogeneous items?

Discussion 1.3: Network psychometrics views symptoms as causally connected rather than caused by a latent factor. Could AI capabilities be "network-like" rather than "factor-like"?

What evidence would distinguish these views? How would it change how we interpret benchmark scores?

Discussion 1.4: S. Truong, Tu, et al. (2025) show that IRT ability θ scales linearly with log(FLOP) during pre-training, and that this relationship enables cross-benchmark transfer of ability estimates. What does this imply about the nature of the latent construct θ? Is it a stable property of the model, or an artifact of the IRT parameterization? Under what conditions would cross-benchmark transfer of θ fail?

3.8 Bibliographic Notes

The probabilistic models we use today emerged from over a century of work across psychology, education, economics, and statistics. Understanding this history illuminates why certain models dominate and what problems they were designed to solve.

The story begins with L.L. Thurstone at the University of Chicago. In 1927, Thurstone proposed a model for how people make pairwise comparisons: the Law of Comparative Judgment (Thurstone 1927). Thurstone's insight was that subjective quantities (preferences, attitudes, perceived stimuli) could be placed on a numerical scale by analyzing patterns of pairwise comparisons. If we ask many people whether stimulus A is greater than stimulus B, and record the proportion who say yes, we can infer the underlying scale values. Acording to Thurstone's model, each stimulus i has a true scale value θ_i. When comparing stimuli i and j, each is perceived with Gaussian noise: $\tilde{\theta}_i \sim \mathcal{N}(\theta_i, \sigma^2), \quad \tilde{\theta}_j \sim \mathcal{N}(\theta_j, \sigma^2)$. The probability that i is judged greater than j is:

$$P(i \succ j) = \Phi\left(\frac{\theta_i - \theta_j}{\sqrt{2}\sigma}\right)$$

where Φ is the standard normal CDF. Thurstone's method showed that subjective quantities could be measured scientifically. The same mathematics now underlies how we rank AI systems from human preferences.

In 1952, Ralph Bradley and Milton Terry developed a model for ranking from paired comparisons in the context of incomplete block designs (Bradley and Milton E. Terry 1952b). Their model:

$$P(i \succ j) = \frac{\pi_i}{\pi_i + \pi_j}$$

where $\pi_i > 0$ are "worth" parameters. With $\theta_i = \log \pi_i$, this becomes the familiar logistic form. In 1959, R. Duncan Luce provided an axiomatic foundation through Choice Axiom: the ratio of choice probabilities for two alternatives should be independent of what other alternatives are available. This axiom leads directly to the Bradley-Terry/logistic model.

In 1960, Danish mathematician Georg Rasch published "Probabilistic Models for Some Intelligence and Attainment Tests," which introduced what we now call the Rasch model (Rasch

1960). Rasch's contribution was the philosophical framework of specific objectivity: the requirement that person and item parameters must be separable. Rasch's work was introduced to the United States by Benjamin Wright at the University of Chicago, who heard Rasch lecture in 1960. Wright became the leading advocate for Rasch measurement in the English-speaking world, founding the MESA (Measurement, Evaluation, Statistical Analysis) program and the journal Rasch Measurement Transactions.

The view of measurement as a homomorphism from an empirical relational structure onto the numbers is *representational measurement theory*, given its canonical treatment in Krantz, Luce, Suppes, and Tversky's Foundations of Measurement (Krantz et al. 1971); Stevens (1946) provides the older scale-typology (nominal/ordinal/interval/ratio) that it refines. Additive conjoint measurement originates with Luce and Tukey's "Simultaneous Conjoint Measurement" (Luce and Tukey 1964), which showed that the cancellation axioms on an ordered two-factor structure suffice to construct interval scales without a prior unit. That the Rasch model realizes this structure was argued by Brogden (1977) and Perline et al. (1979). The inference is contested: Michell (2008) and Kyngdon (2008) dispute whether reading a *deterministic* conjoint order off *stochastic* responses is licensed, a debate the chapter treats as open. The order-restricted, probabilistic formulations that make the axioms empirically testable—rather than assumed—are developed in Karabatsos (2001) and Scheiblechner (1995); these are the tools behind this chapter's claim that AI evaluation, by resampling at temperature > 0, can *test* additive conjoint structure where human testing could only posit it.

In the 2010s, these methods reached AI evaluation—the application of IRT to benchmark analysis (Polo et al. 2024), multidimensional factor models for AI capabilities (this textbook), and the Chatbot Arena's use of Elo ratings to rank language models (Chiang et al. 2024).

4 Learning

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Derive** the log-likelihood function for the Rasch model and explain the role of person and item parameters.
2. **Implement** maximum likelihood estimation (MLE) for IRT models using gradient ascent and L-BFGS optimization.
3. **Explain** the identifiability problem in IRT and describe standard solutions (sum-to-zero, fixed anchor).
4. **Distinguish** between joint MLE, conditional MLE, and marginal MLE, and articulate when each is appropriate.
5. **Implement** the Expectation-Maximization (EM) algorithm for Rasch model estimation and explain the E-step and M-step.
6. **Describe** Bayesian inference for IRT models and specify appropriate priors for ability and item parameters.
7. **Implement** MAP estimation and MCMC sampling for IRT models.
8. **Explain** regularization in IRT as a Bayesian prior and apply cross-validation for hyperparameter selection.
9. **Apply** MLE and Bayesian methods to real AI benchmark data and compare their efficiency.

Suggested Lecture Plan

This chapter can be covered in **3 lectures** (75-90 minutes each):

Lecture 1: Foundations of Estimation

- Why learning matters for AI measurement (15 min)
- Likelihood and log-likelihood for Rasch model (20 min)
- Gradient derivation and interpretation (20 min)
- Hands-on: MLE with synthetic data (20 min)

Lecture 2: Advanced Estimation Methods

- Identifiability and conditional vs marginal MLE (20 min)
- EM algorithm for IRT (30 min)
- Hands-on: EM implementation (25 min)

Lecture 3: Bayesian Approaches

- Prior specification for IRT (15 min)
- MAP estimation and MCMC (30 min)
- Regularization and cross-validation (15 min)

Notation

This chapter introduces estimation notation: $\ell(\theta, \beta)$ (log-likelihood), $\hat{\theta}_{\text{MLE}}$ and $\hat{\theta}_{\text{MAP}}$ (point estimates), $\pi(\theta)$ (priors), and η (learning rate). See the notation appendix[1] for the complete notation reference.

[1] notation.qmd

Video Overview

A visual tour of the key concepts in this chapter — from maximum likelihood estimation and the EM algorithm to Bayesian inference and generalization.

../animations/ch2/chapter2_narrated.mp4

4.1 Why Learning Matters for AI Measurement

Chapter 3 introduced the measurement models—Rasch, 2PL, Bradley-Terry—that describe how latent abilities generate observed responses. But knowing the *form* of a model is not enough. To actually *use* these models for AI evaluation, we must estimate their parameters from data. This chapter develops estimation for the unidimensional IRT models; the multidimensional factor model and the experiments that probe its generalization are estimated and analyzed in Section 7.2 and Section 7.5.

The Central Learning Problem in AI Measurement

Given a response matrix $Y \in \{0, 1\}^{N \times M}$ where $Y_{ij} = 1$ indicates model i answered question j correctly:

$$\text{Find } \hat{\theta}, \hat{\beta} = \arg\max_{\theta, \beta} P(Y \mid \theta, \beta)$$

This optimization problem underlies all psychometric estimation and forms the foundation for trustworthy AI evaluation.

Parameter estimation serves several critical purposes in AI measurement:

1. **Fair comparison**: Calibrated item difficulties allow us to compare models tested on different question subsets. If we know that question A is harder than question B, we can appropriately weight their contributions to the final score.
2. **Uncertainty quantification**: Estimation procedures provide not just point estimates but standard errors, telling us how confident we should be in our measurements.
3. **Adaptive testing**: Once we have calibrated item parameters, we can select the most informative questions for each model, dramatically reducing evaluation costs (see Section 6.2.2 for a full treatment of Computerized Adaptive Testing).

4. **Prediction**: With learned parameters, we can predict how a model will perform on questions it has never seen, enabling efficient evaluation of new benchmarks.

This chapter focuses on **passive learning**: given a fixed dataset, estimate all parameters simultaneously. This includes maximum likelihood estimation (MLE), expectation-maximization (EM), and Bayesian inference. The design of *which* items to include in a benchmark, including active item selection, is the subject of Chapter 6.

4.2 Maximum Likelihood Estimation

Maximum likelihood estimation is the foundation of parameter estimation in IRT. The principle is simple: find the parameter values that make the observed data most probable.

4.2.1 The Likelihood Function

Recall from Chapter 3 that the Rasch model specifies the probability of a correct response as:

$$P(Y_{ij} = 1 \mid \theta_i, \beta_j) = \sigma(\theta_i - \beta_j) = \frac{1}{1 + e^{-(\theta_i - \beta_j)}} \tag{4.1}$$

where θ_i is the ability of model i and β_j is the difficulty of item j.

Under the *local independence* assumption introduced in **?@sec-local-independence**—responses are conditionally independent given the latent parameters—the likelihood of the entire response matrix factorizes into a product of per-observation terms:

$$L(\theta, \beta \mid Y) = \prod_{i=1}^{N} \prod_{j=1}^{M} P(Y_{ij} \mid \theta_i, \beta_j)^{Y_{ij}} [1 - P(Y_{ij} \mid \theta_i, \beta_j)]^{1 - Y_{ij}} \tag{4.2}$$

Taking the logarithm (for computational stability and mathematical convenience):

$$\ell(\theta, \beta) = \sum_{i=1}^{N} \sum_{j=1}^{M} \left[Y_{ij}(\theta_i - \beta_j) - \log(1 + e^{\theta_i - \beta_j}) \right] \tag{4.3}$$

This is the objective function we want to maximize.

4.2.2 Gradient Derivation

To optimize the log-likelihood, we need its gradients. Taking partial derivatives:

$$\frac{\partial \ell}{\partial \theta_i} = \sum_{j=1}^{M} \left[Y_{ij} - \sigma(\theta_i - \beta_j) \right] \tag{4.4}$$

$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i=1}^{N} \left[\sigma(\theta_i - \beta_j) - Y_{ij} \right] \tag{4.5}$$

Intuitive Interpretation of the Gradient

The gradient $\frac{\partial \ell}{\partial \theta_i} = \sum_j [Y_{ij} - P_{ij}]$ has a beautiful interpretation:

- Y_{ij} is the **observed** response (0 or 1)
- $P_{ij} = \sigma(\theta_i - \beta_j)$ is the **predicted** probability

The gradient is simply the sum of **residuals**: observed minus predicted. If model i performs better than expected (more correct answers than predicted), the residuals are positive, and we increase θ_i. If it performs worse than expected, we decrease θ_i. This is the essence of gradient ascent.

4.2.3 The Identifiability Problem

Before implementing the optimizer, we must address a subtlety that would otherwise cause the iterates to drift indefinitely without improving fit.

The Identifiability Problem

The Rasch model has a fundamental identifiability issue: if we add a constant c to all abilities and all difficulties, the likelihood is unchanged:

$$P(Y_{ij} = 1 \mid \theta_i + c, \beta_j + c) = \sigma((\theta_i + c) - (\beta_j + c)) = \sigma(\theta_i - \beta_j)$$

The parameters are only identified up to an additive constant. This means infinitely many parameter values produce the same likelihood.

Common Solutions:

1. **Sum-to-zero constraint**: Set $\sum_i \theta_i = 0$ or $\sum_j \beta_j = 0$
2. **Fixed anchor**: Set one parameter (e.g., $\beta_1 = 0$) as reference
3. **Prior constraint**: Use Bayesian priors centered at zero

For AI benchmarks, we typically use sum-to-zero: a model with $\theta = 0$ has "average" ability relative to the calibration sample.

We will adopt the sum-to-zero constraint below, re-centering $\hat{\theta}$ and $\hat{\beta}$ after each gradient step so the iterates stay anchored.

4.2.4 Implementation with Gradient Ascent

Let us implement MLE via gradient ascent on synthetic data. First, we generate a response matrix from known parameters:

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: synthetic-data
#| autorun: true
#| fig-cap: "Synthetic response matrix generated from known Rasch model parameters."

import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    """Numerically stable sigmoid function."""
    return np.where(x >= 0,
                    1 / (1 + np.exp(-x)),
                    np.exp(x) / (1 + np.exp(x)))

# Set seed for reproducibility
np.random.seed(42)

# True parameters
N, M = 100, 50  # 100 models, 50 questions
theta_true = np.random.normal(0, 1, N)  # True abilities
beta_true = np.random.normal(0, 1.5, M)  # True difficulties
```

```

# Generate response matrix via Rasch model
prob_matrix = sigmoid(theta_true[:, None] - beta_true[None, :])
Y = (np.random.random((N, M)) < prob_matrix).astype(int)

print(f"Response matrix shape: {Y.shape}")
print(f"Overall accuracy: {Y.mean():.3f}")
print(f"Model accuracies range: [{Y.mean(axis=1).min():.3f},
 ↪  {Y.mean(axis=1).max():.3f}]")
print(f"Item difficulties range: [{Y.mean(axis=0).min():.3f},
 ↪  {Y.mean(axis=0).max():.3f}]")

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(6, 2))

# Raw response matrix
axes[0].imshow(Y, aspect='auto', cmap='Blues')
axes[0].set_xlabel('Questions')
axes[0].set_ylabel('Models')
axes[0].set_title('Raw Response Matrix')

# Sorted by ability and difficulty
row_order = np.argsort(Y.mean(axis=1))[::-1]
col_order = np.argsort(Y.mean(axis=0))[::-1]
Y_sorted = Y[row_order][:, col_order]
axes[1].imshow(Y_sorted, aspect='auto', cmap='Blues')
axes[1].set_xlabel('Questions (sorted by difficulty)')
axes[1].set_ylabel('Models (sorted by ability)')
axes[1].set_title('Sorted Response Matrix')

plt.tight_layout()
plt.show()
```

Next, we define helper functions for the log-likelihood and its gradients derived in Section 4.2.2:

```
#| label: rasch-helpers
#| autorun: true

def rasch_log_likelihood(theta, beta, Y):
    """Compute Rasch model log-likelihood."""
    logits = theta[:, None] - beta[None, :]
    ll = (Y * logits - np.log(1 + np.exp(np.clip(logits, -500, 500)))).sum()
    return ll

def rasch_gradients(theta, beta, Y):
    """Compute gradients for theta and beta."""
    P = sigmoid(theta[:, None] - beta[None, :])
```

```
    grad_theta = (Y - P).sum(axis=1)
    grad_beta = (P - Y).sum(axis=0)
    return grad_theta, grad_beta
```

With the helpers in place, we run the gradient ascent loop. At each step we take a gradient step and then apply the sum-to-zero re-centering described in Section 4.2.3:

```
#| label: mle-gradient-ascent
#| autorun: true
#| fig-cap: "Convergence of gradient ascent for Rasch model MLE."

# Initialize parameters at zero
theta_hat = np.zeros(N)
beta_hat = np.zeros(M)

# Gradient ascent
learning_rate = 0.01
n_iterations = 500
ll_history = []

for iteration in range(n_iterations):
    # Compute gradients
    grad_theta, grad_beta = rasch_gradients(theta_hat, beta_hat, Y)

    # Update parameters
    theta_hat = theta_hat + learning_rate * grad_theta
    beta_hat = beta_hat + learning_rate * grad_beta

    # Re-center for identification (sum-to-zero constraint)
    theta_hat = theta_hat - theta_hat.mean()
    beta_hat = beta_hat - beta_hat.mean()

    # Track log-likelihood
    ll = rasch_log_likelihood(theta_hat, beta_hat, Y)
    ll_history.append(ll)

# Plot convergence
fig, axes = plt.subplots(1, 3, figsize=(6, 2))

# Convergence curve
axes[0].plot(ll_history)
axes[0].set_xlabel('Iteration')
axes[0].set_ylabel('Log-likelihood')
axes[0].set_title('Gradient Ascent Convergence')
axes[0].grid(True, alpha=0.3)

# Compare ability estimates to true values
theta_true_centered = theta_true - theta_true.mean()
```

```
axes[1].scatter(theta_true_centered, theta_hat, alpha=0.6)
axes[1].plot([-3, 3], [-3, 3], 'k--', alpha=0.5, label='y=x')
axes[1].set_xlabel('True ability (centered)')
axes[1].set_ylabel('Estimated ability')
axes[1].set_title('Recovery of Abilities')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

# Compare difficulty estimates to true values
beta_true_centered = beta_true - beta_true.mean()
axes[2].scatter(beta_true_centered, beta_hat, alpha=0.6, color='orange')
axes[2].plot([-4, 4], [-4, 4], 'k--', alpha=0.5, label='y=x')
axes[2].set_xlabel('True difficulty (centered)')
axes[2].set_ylabel('Estimated difficulty')
axes[2].set_title('Recovery of Difficulties')
axes[2].legend()
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Correlation with true values
corr_theta = np.corrcoef(theta_true_centered, theta_hat)[0, 1]
corr_beta = np.corrcoef(beta_true_centered, beta_hat)[0, 1]
print(f"Correlation with true abilities: {corr_theta:.4f}")
print(f"Correlation with true difficulties: {corr_beta:.4f}")
```

4.2.5 L-BFGS Optimization

While gradient ascent is intuitive, quasi-Newton methods like L-BFGS converge much faster by approximating second-order information. L-BFGS is conventionally formulated as a minimization routine, so we negate the log-likelihood and minimize the resulting objective:

```
#| label: lbfgs-optimization
#| autorun: true
#| fig-cap: "L-BFGS achieves faster convergence than gradient ascent."

from scipy.optimize import minimize

def negative_log_likelihood(params, Y):
    """Negative log-likelihood (for minimization)."""
    N, M = Y.shape
    theta = params[:N]
    beta = params[N:]

    logits = theta[:, None] - beta[None, :]
    nll = -(Y * logits - np.log(1 + np.exp(np.clip(logits, -500, 500)))).sum()
```

```
    return nll

def gradient(params, Y):
    """Gradient of negative log-likelihood."""
    N, M = Y.shape
    theta = params[:N]
    beta = params[N:]

    P = sigmoid(theta[:, None] - beta[None, :])
    grad_theta = -(Y - P).sum(axis=1)
    grad_beta = -(P - Y).sum(axis=0)

    return np.concatenate([grad_theta, grad_beta])

# Initial parameters
params0 = np.zeros(N + M)

# L-BFGS optimization
result = minimize(
    negative_log_likelihood,
    params0,
    args=(Y,),
    jac=gradient,
    method='L-BFGS-B',
    options={'maxiter': 200, 'disp': False}
)

theta_lbfgs = result.x[:N]
beta_lbfgs = result.x[N:]

# Center for comparison
theta_lbfgs = theta_lbfgs - theta_lbfgs.mean()
beta_lbfgs = beta_lbfgs - beta_lbfgs.mean()

print(f"L-BFGS converged: {result.success}")
print(f"Final log-likelihood: {-result.fun:.2f}")
print(f"Iterations: {result.nit}")

# Compare to gradient ascent
print(f"\nCorrelation with GA estimates:")
print(f"  Abilities: {np.corrcoef(theta_hat, theta_lbfgs)[0,1]:.6f}")
print(f"  Difficulties: {np.corrcoef(beta_hat, beta_lbfgs)[0,1]:.6f}")
```

4.3 Joint, Conditional, and Marginal MLE

The MLE approach we have discussed so far is called *joint maximum likelihood estimation* (JMLE). It treats both person parameters θ and item parameters β as fixed unknowns to be estimated. However, JMLE has theoretical limitations that motivate alternative approaches.

4.3.1 Joint MLE (JMLE)

JMLE simultaneously estimates all parameters by maximizing Equation 4.3. While intuitive, JMLE suffers from the *incidental parameter problem* (Neyman and Scott 1948): as the number of items M remains fixed and the number of persons N grows, the item parameter estimates $\hat{\beta}$ are inconsistent—they do not converge to the true values. Haberman (1977) proves this formally for the Rasch model; Lancaster (2000) surveys how the problem generalizes across statistics and econometrics.

> **Intuition: Why More Data Does Not Fix the Bias**
>
> For a standard estimation problem, more data means better estimates: the law of large numbers washes out idiosyncratic noise and $\hat{\beta}$ converges to the truth. One might hope the same holds here: as we add more test takers, the sample information about each item β_j grows without bound, so $\hat{\beta}_j$ should become unbiased.
>
> The catch is that each new test taker also brings a new parameter θ_i to estimate. Unlike β_j, the person parameter θ_i is *only* estimated from that person's M responses — adding more test takers does **not** give us more information about any existing θ_i. These per-person parameters are "incidental": their number grows with N, but their precision does not.
>
> When we plug the noisy $\hat{\theta}_i$ back into the likelihood to update $\hat{\beta}_j$, the bias in the person estimates leaks into the item estimates, and the two biases do not cancel. As a result, $\hat{\beta}_j$ is biased by a factor that depends only on M — more persons cannot rescue us; only more *items per person* can.

For AI benchmarks with many items (typically $M > 100$), this bias is small in practice. But for smaller tests, JMLE can be problematic, which motivates the conditional and marginal approaches below.

4.3.2 Conditional MLE (CMLE)

Georg Rasch discovered an elegant solution to the incidental parameter problem. For the Rasch model specifically, the sum score $S_i = \sum_j Y_{ij}$ is a *sufficient statistic* for θ_i. This means all information about θ_i in the data Y_i is captured by S_i.

By conditioning on the sufficient statistics, we can eliminate the person parameters entirely:

$$P(Y_i \mid S_i, \beta) = \frac{\exp(-\sum_j Y_{ij}\beta_j)}{\gamma_{S_i}(\beta)} \tag{4.6}$$

where $\gamma_r(\beta) = \sum_{A:|A|=r} \exp(-\sum_{j\in A} \beta_j)$ is the elementary symmetric function of order r, summing over all subsets A of items of size r.

The conditional likelihood depends only on β, so we can estimate item parameters without any person parameters. This produces consistent estimates of β regardless of how N grows.

Rasch's Insight

The sufficiency of sum scores is unique to the Rasch model. For the 2PL or 3PL models, sum scores are not sufficient, and CMLE cannot be applied. This mathematical property is one reason the Rasch model holds special status in measurement theory.

4.3.3 Marginal MLE (MMLE)

An alternative approach is to treat person parameters as random variables from a population distribution:

$$\theta_i \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2)$$

The marginal likelihood integrates out the person parameters:

$$L(\beta) = \prod_{i=1}^{N} \int P(Y_i \mid \theta, \beta)\, p(\theta)\, d\theta \tag{4.7}$$

This approach:

- Treats item parameters as fixed and person parameters as random
- Produces consistent estimates of β as $N \to \infty$
- Naturally extends to more complex IRT models (2PL, 3PL)
- Forms the basis for the EM algorithm (next section)

Comparison of MLE Approaches

Method	Person Parameters	Item Parameters	Consistency	Applicability
JMLE	Estimated directly	Estimated directly	Inconsistent for fixed M	Any IRT model
CMLE	Conditioned out	Consistent	Consistent	Rasch only
MMLE	Integrated out	Consistent	Consistent	Any IRT model

For AI benchmarks with many questions ($M > 100$), JMLE works well in practice. For smaller tests or when statistical properties are important, CMLE or MMLE is preferred.

4.4 The EM Algorithm

The Expectation-Maximization (EM) algorithm is a general method for maximum likelihood estimation with latent variables. In IRT, the latent variables are the person abilities θ.

4.4.1 The EM Framework

The EM algorithm iterates between two steps:

E-step (Expectation): Compute the expected value of the complete-data log-likelihood, given the observed data and current parameter estimates:

$$Q(\beta \mid \beta^{(t)}) = \mathbb{E}_{\theta|Y,\beta^{(t)}} \left[\log P(Y, \theta \mid \beta)\right]$$

M-step (Maximization): Find the parameter values that maximize the expected log-likelihood:

$$\beta^{(t+1)} = \arg\max_{\beta} Q(\beta \mid \beta^{(t)})$$

The EM algorithm guarantees that the marginal likelihood increases (or stays the same) at each iteration, converging to a local maximum.

4.4.2 EM for the Rasch Model

For the Rasch model with a standard normal prior on abilities, the EM algorithm takes a specific form:

E-step: For each person i, compute the posterior distribution of θ_i given their responses Y_i and current item parameters $\beta^{(t)}$:

$$p(\theta_i \mid Y_i, \beta^{(t)}) \propto p(Y_i \mid \theta_i, \beta^{(t)}) \cdot p(\theta_i)$$

This posterior is not available in closed form, so we use numerical integration (Gauss-Hermite quadrature).

M-step: Update each item parameter by solving:

$$\sum_{i=1}^{N} \mathbb{E}_{\theta_i}[\sigma(\theta_i - \beta_j)] = \sum_{i=1}^{N} Y_{ij}$$

The left side is the expected number of correct responses to item j; the right side is the observed number. We equate these.

```
#| label: em-algorithm
#| autorun: true
#| fig-cap: "EM algorithm convergence for Rasch model estimation."

from numpy.polynomial.hermite import hermgauss
```

```

def em_rasch(Y, n_iterations=50, n_quadrature=21, verbose=True):
    """
    EM algorithm for Rasch model using Gauss-Hermite quadrature.

    Parameters
    ----------
    Y : ndarray (N, M)
        Binary response matrix
    n_iterations : int
        Number of EM iterations
    n_quadrature : int
        Number of quadrature points
    verbose : bool
        Print progress

    Returns
    -------
    theta_hat : ndarray (N,)
        Estimated abilities (posterior means)
    beta_hat : ndarray (M,)
        Estimated difficulties
    ll_history : list
        Marginal log-likelihood at each iteration
    """
    N, M = Y.shape

    # Initialize item difficulties
    beta = np.zeros(M)

    # Gauss-Hermite quadrature points and weights
    # These approximate the integral over theta ~ N(0, 1)
    nodes, weights = hermgauss(n_quadrature)
    nodes = nodes * np.sqrt(2)  # Scale for standard normal
    weights = weights / np.sqrt(np.pi)  # Normalize

    ll_history = []

    for iteration in range(n_iterations):
        # E-step: Compute posterior distributions over theta
        # P(theta | Y_i, beta) for each person at each quadrature point

        # Compute log-likelihood at each quadrature point for each person
        # log P(Y_i | theta_q, beta) for all i, q
        log_L = np.zeros((N, n_quadrature))
        for q, theta_q in enumerate(nodes):
            logits = theta_q - beta  # (M,)
            # log P(Y_i | theta_q) = sum_j [Y_ij * logit_j - log(1 + exp(logit_j))]
            log_probs = Y * logits - np.log(1 + np.exp(np.clip(logits, -500, 500)))
```

```
            log_L[:, q] = log_probs.sum(axis=1)

        # Compute posterior weights: P(theta_q | Y_i, beta)  P(Y_i | theta_q) *
↪ P(theta_q)
        # The weights already incorporate P(theta_q) from Gauss-Hermite
        log_posterior = log_L + np.log(weights + 1e-300)

        # Normalize to get posterior probabilities
        log_posterior_max = log_posterior.max(axis=1, keepdims=True)
        posterior = np.exp(log_posterior - log_posterior_max)
        posterior = posterior / posterior.sum(axis=1, keepdims=True)

        # Expected ability for each person (posterior mean)
        E_theta = (posterior * nodes).sum(axis=1)

        # M-step: Update beta
        # For each item j, solve: sum_i E[P(Y_ij=1 | theta_i)] = sum_i Y_ij
        for j in range(M):
            # Expected probability at each quadrature point
            for _ in range(5):  # Newton-Raphson iterations
                E_prob_j = np.zeros(N)
                E_deriv_j = np.zeros(N)
                for q, theta_q in enumerate(nodes):
                    p_q = sigmoid(theta_q - beta[j])
                    E_prob_j += posterior[:, q] * p_q
                    E_deriv_j += posterior[:, q] * p_q * (1 - p_q)

                # Newton-Raphson update
                residual = E_prob_j.sum() - Y[:, j].sum()
                hessian = -E_deriv_j.sum()
                if abs(hessian) > 1e-10:
                    beta[j] = beta[j] - residual / hessian

        # Center beta for identification
        beta = beta - beta.mean()

        # Compute marginal log-likelihood for monitoring
        ll = (log_posterior_max.flatten() +
              np.log(np.exp(log_L - log_posterior_max) @ weights + 1e-300)).sum()
        ll_history.append(ll)

        if verbose and (iteration + 1) % 10 == 0:
            print(f"Iteration {iteration + 1}: LL = {ll:.2f}")

    # Final E-step to get ability estimates
    log_L = np.zeros((N, n_quadrature))
    for q, theta_q in enumerate(nodes):
        logits = theta_q - beta
        log_probs = Y * logits - np.log(1 + np.exp(np.clip(logits, -500, 500)))
```

```
        log_L[:, q] = log_probs.sum(axis=1)

    log_posterior = log_L + np.log(weights + 1e-300)
    log_posterior_max = log_posterior.max(axis=1, keepdims=True)
    posterior = np.exp(log_posterior - log_posterior_max)
    posterior = posterior / posterior.sum(axis=1, keepdims=True)
    theta_hat = (posterior * nodes).sum(axis=1)

    return theta_hat, beta, ll_history

# Run EM algorithm
theta_em, beta_em, ll_em = em_rasch(Y, n_iterations=50)

# Plot results
fig, axes = plt.subplots(1, 3, figsize=(6, 2))

# Convergence
axes[0].plot(ll_em)
axes[0].set_xlabel('Iteration')
axes[0].set_ylabel('Marginal Log-likelihood')
axes[0].set_title('EM Algorithm Convergence')
axes[0].grid(True, alpha=0.3)

# Ability recovery
axes[1].scatter(theta_true_centered, theta_em, alpha=0.6)
axes[1].plot([-3, 3], [-3, 3], 'k--', alpha=0.5)
axes[1].set_xlabel('True ability (centered)')
axes[1].set_ylabel('EM estimate')
axes[1].set_title('Ability Recovery (EM)')
axes[1].grid(True, alpha=0.3)

# Difficulty recovery
axes[2].scatter(beta_true_centered, beta_em, alpha=0.6, color='orange')
axes[2].plot([-4, 4], [-4, 4], 'k--', alpha=0.5)
axes[2].set_xlabel('True difficulty (centered)')
axes[2].set_ylabel('EM estimate')
axes[2].set_title('Difficulty Recovery (EM)')
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"Correlation (abilities): {np.corrcoef(theta_true_centered,
 ↪  theta_em)[0,1]:.4f}")
print(f"Correlation (difficulties): {np.corrcoef(beta_true_centered,
 ↪  beta_em)[0,1]:.4f}")
```

4.5 Bayesian Inference

Bayesian inference provides an alternative to maximum likelihood that naturally incorporates prior information and quantifies uncertainty. Instead of finding a single point estimate, we characterize the entire posterior distribution over parameters.

4.5.1 Prior Specification

The first step in Bayesian inference is specifying prior distributions that encode our beliefs before seeing the data:

Standard Priors for IRT

For abilities (persons/models):

$$\theta_i \sim \mathcal{N}(0, \sigma_\theta^2), \quad \sigma_\theta = 1 \text{ (standard choice)}$$

For difficulties (items/questions):

$$\beta_j \sim \mathcal{N}(0, \sigma_\beta^2), \quad \sigma_\beta = 1\text{-}2 \text{ (depending on expected range)}$$

For discrimination (2PL model):

$$\beta_{j2} \sim \text{LogNormal}(0, 0.5) \text{ or } \beta_{j2} \sim \text{Gamma}(2, 0.5)$$

These priors are **weakly informative**: they regularize estimates without dominating the data. They encode the belief that most abilities and difficulties are within a few units of zero, which is appropriate when the scale is defined by convention.

4.5.2 Hierarchical Priors and Partial Pooling

The priors above are *fixed*: we choose σ_β once and apply it to every item. But AI evaluations are *nested*—items belong to benchmarks, benchmarks to suites or domains—and items within a benchmark tend to share difficulty characteristics. A **hierarchical prior** turns the fixed prior into one whose hyperparameters are *learned* from the data, letting items borrow strength within their group.

Definition: Hierarchical IRT Model

Consider item i nested within benchmark j. A hierarchical extension of the Rasch model specifies:

$$\text{logit}\, P(Y_{ij} = 1 \mid \theta, b_{ij}) = \theta - b_{ij}$$

where item difficulties are drawn from a benchmark-level prior:

$$b_{ij} \sim \mathcal{N}(\mu_j, \sigma_j^2)$$

The benchmark means μ_j may themselves follow a domain-level distribution $\mu_j \sim \mathcal{N}(\mu_0, \tau^2)$, creating a three-level hierarchy: items within benchmarks within domains. The same hierarchical extension applies to 2PL, 3PL, and factor models.

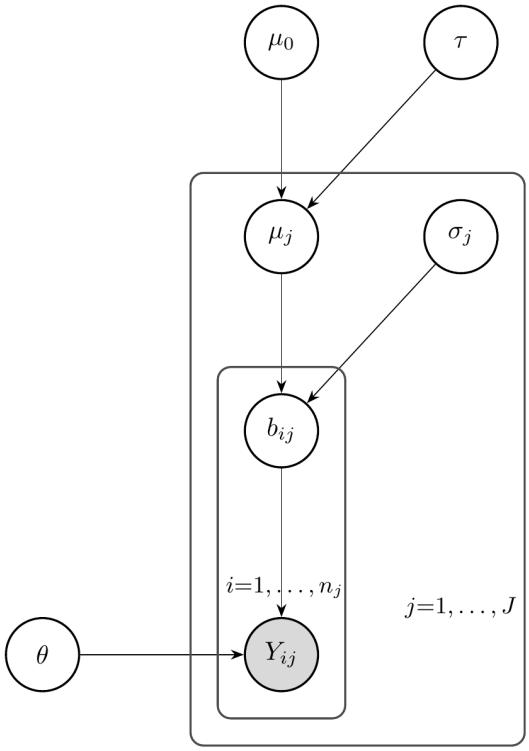


Figure 4.1
Plate diagram for the hierarchical IRT model. Item difficulties b_{ij} are drawn from benchmark-level distributions parameterized by μ_j and σ_j, which may themselves be drawn from domain-level hyperparameters μ_0 and τ. Nested plates reflect the hierarchical data structure.

Estimating this model performs **partial pooling**, interpolating between two extremes. *No pooling* estimates each benchmark's difficulties independently—noisy for small benchmarks. *Complete pooling* uses one global difficulty distribution—ignoring real between-benchmark differences. The hierarchical prior sits between them: an item's difficulty is shrunk toward its benchmark mean μ_j by an amount that depends on how sharply the data pin it down, and the benchmark means are themselves shrunk toward the global mean μ_0. Items in small or noisy benchmarks borrow strength from their group, and because the hyperparameters σ_j, τ are learned (by MAP or MCMC; Section 4.5.4, Section 4.5.5), the *amount* of shrinkage is set by the data rather than fixed in advance. This is the same move as L2 regularization (Section 4.6.1)—a Gaussian prior on the parameters—but with the prior's scale estimated rather than chosen. Treating all items as exchangeable under a single global prior instead conflates within-benchmark and between-benchmark variation and yields estimates that may not generalize beyond the items

observed (Luettgau et al. 2025).

Hierarchical Structure in AI Evaluation

Modern AI evaluations exhibit natural hierarchy at multiple levels:

- **MMLU**: 15,908 items → 57 subjects → 4 domains (humanities, social sciences, STEM, other)
- **GAIA**: agentic tasks → 3 difficulty levels → capability domain
- **Coding benchmarks**: problems → benchmarks (HumanEval, MBPP, DS-1000) → coding capability

Modeling these levels separates the sources of variation at each, which supports principled generalization from the benchmarks actually tested to the broader construct they are intended to measure.

4.5.3 Posterior Computation

Bayes' theorem gives us the posterior distribution:

$$p(\theta, \beta \mid Y) \propto p(Y \mid \theta, \beta) \cdot p(\theta) \cdot p(\beta) \tag{4.8}$$

The posterior combines the likelihood (data) with the priors (beliefs). Unfortunately, this posterior is not available in closed form—we need computational methods.

4.5.4 MAP Estimation

The simplest Bayesian approach is *maximum a posteriori* (MAP) estimation, which finds the mode of the posterior:

$$\hat{\theta}_{\text{MAP}}, \hat{\beta}_{\text{MAP}} = \arg\max_{\theta,\beta} \left[\ell(\theta, \beta \mid Y) + \log p(\theta) + \log p(\beta)\right] \tag{4.9}$$

With Gaussian priors, this is equivalent to L2-regularized MLE:

$$\hat{\theta}_{\text{MAP}}, \hat{\beta}_{\text{MAP}} = \arg\max_{\theta,\beta} \left[\ell(\theta, \beta) - \frac{1}{2\sigma_\theta^2}\sum_i \theta_i^2 - \frac{1}{2\sigma_\beta^2}\sum_j \beta_j^2\right]$$

```
#| label: map-estimation
#| autorun: true
#| fig-cap: "Comparison of MLE and MAP estimates showing Bayesian shrinkage."

def map_objective(params, Y, sigma_theta=1.0, sigma_beta=1.5):
    """Negative log-posterior (to minimize)."""
    N, M = Y.shape
    theta = params[:N]
```

```
    beta = params[N:]

    # Log-likelihood
    logits = theta[:, None] - beta[None, :]
    ll = (Y * logits - np.log(1 + np.exp(np.clip(logits, -500, 500)))).sum()

    # Log-prior (Gaussian)
    log_prior_theta = -0.5 * (theta**2 / sigma_theta**2).sum()
    log_prior_beta = -0.5 * (beta**2 / sigma_beta**2).sum()

    return -(ll + log_prior_theta + log_prior_beta)

def map_gradient(params, Y, sigma_theta=1.0, sigma_beta=1.5):
    """Gradient of negative log-posterior."""
    N, M = Y.shape
    theta = params[:N]
    beta = params[N:]

    P = sigmoid(theta[:, None] - beta[None, :])
    grad_theta = -(Y - P).sum(axis=1) + theta / sigma_theta**2
    grad_beta = -(P - Y).sum(axis=0) + beta / sigma_beta**2

    return np.concatenate([grad_theta, grad_beta])

# MAP estimation
params0 = np.zeros(N + M)
result_map = minimize(
    map_objective, params0, args=(Y,),
    jac=map_gradient,
    method='L-BFGS-B',
    options={'maxiter': 200}
)

theta_map = result_map.x[:N]
beta_map = result_map.x[N:]

# Center for comparison
theta_map = theta_map - theta_map.mean()
beta_map = beta_map - beta_map.mean()

# Compare MLE vs MAP
fig, axes = plt.subplots(1, 2, figsize=(6, 2))

# Abilities
axes[0].scatter(theta_true_centered, theta_lbfgs, alpha=0.5, label='MLE', s=30)
axes[0].scatter(theta_true_centered, theta_map, alpha=0.5, label='MAP', s=30)
axes[0].plot([-3, 3], [-3, 3], 'k--', alpha=0.5)
axes[0].set_xlabel('True ability')
axes[0].set_ylabel('Estimated ability')
```

```
axes[0].set_title('Ability Estimates: MLE vs MAP')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Difficulties
axes[1].scatter(beta_true_centered, beta_lbfgs, alpha=0.5, label='MLE', s=30)
axes[1].scatter(beta_true_centered, beta_map, alpha=0.5, label='MAP', s=30)
axes[1].plot([-4, 4], [-4, 4], 'k--', alpha=0.5)
axes[1].set_xlabel('True difficulty')
axes[1].set_ylabel('Estimated difficulty')
axes[1].set_title('Difficulty Estimates: MLE vs MAP')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Shrinkage demonstration
print("Shrinkage effect (standard deviations):")
print(f"  MLE abilities: {theta_lbfgs.std():.3f}, MAP abilities:
 ↪  {theta_map.std():.3f}")
print(f"  MLE difficulties: {beta_lbfgs.std():.3f}, MAP difficulties:
 ↪  {beta_map.std():.3f}")
```

Bayesian Shrinkage

Notice that MAP estimates have smaller variance than MLE estimates. This is **shrinkage** toward the prior mean (zero).

For extreme scores—models that answer all questions correctly or incorrectly—MLE gives infinite or very large estimates. MAP regularizes these to finite, sensible values. This is crucial for AI benchmarks where some models may achieve near-perfect scores on easy subsets.

The amount of shrinkage is controlled by the prior variance: smaller σ^2 means stronger shrinkage toward zero.

4.5.5 MCMC Sampling

To characterize the full posterior distribution (not just its mode), we use Markov Chain Monte Carlo (MCMC) sampling. The Metropolis-Hastings algorithm is a simple but effective approach:

```
#| label: mcmc-sampling
#| autorun: true
#| fig-cap: "MCMC trace plots and posterior distributions for selected parameters."

def log_posterior(theta, beta, Y, sigma_theta=1.0, sigma_beta=1.5):
    """Compute log-posterior (up to normalizing constant)."""
```

```
    logits = theta[:, None] - beta[None, :]
    ll = (Y * logits - np.log(1 + np.exp(np.clip(logits, -500, 500)))).sum()
    log_prior = -0.5 * ((theta**2).sum() / sigma_theta**2 +
                        (beta**2).sum() / sigma_beta**2)
    return ll + log_prior

def metropolis_hastings_rasch(Y, n_samples=2000, n_warmup=500,
                              proposal_sd=0.05, thin=2, verbose=True):
    """
    Metropolis-Hastings sampler for Rasch model.

    Uses a random-walk proposal for all parameters jointly.
    """
    N, M = Y.shape

    # Initialize at MAP estimate
    theta = theta_map.copy()
    beta = beta_map.copy()

    # Storage for samples
    n_stored = n_samples // thin
    theta_samples = np.zeros((n_stored, N))
    beta_samples = np.zeros((n_stored, M))

    current_lp = log_posterior(theta, beta, Y)
    n_accept = 0
    sample_idx = 0

    total_iterations = n_warmup + n_samples

    for s in range(total_iterations):
        # Propose new theta (random walk)
        theta_prop = theta + np.random.normal(0, proposal_sd, N)
        theta_prop = theta_prop - theta_prop.mean()  # Maintain centering

        # Propose new beta (random walk)
        beta_prop = beta + np.random.normal(0, proposal_sd, M)
        beta_prop = beta_prop - beta_prop.mean()  # Maintain centering

        # Compute acceptance probability
        prop_lp = log_posterior(theta_prop, beta_prop, Y)
        log_alpha = prop_lp - current_lp

        # Accept or reject
        if np.log(np.random.random()) < log_alpha:
            theta = theta_prop
            beta = beta_prop
            current_lp = prop_lp
            if s >= n_warmup:
```

```
                n_accept += 1

        # Store sample (after warmup, with thinning)
        if s >= n_warmup and (s - n_warmup) % thin == 0:
            theta_samples[sample_idx] = theta
            beta_samples[sample_idx] = beta
            sample_idx += 1

    acceptance_rate = n_accept / n_samples
    if verbose:
        print(f"Acceptance rate: {acceptance_rate:.3f}")

    return theta_samples, beta_samples, acceptance_rate

# Run MCMC
np.random.seed(123)
theta_samples, beta_samples, acc_rate = metropolis_hastings_rasch(
    Y, n_samples=4000, n_warmup=1000, proposal_sd=0.03, thin=2
)

# Visualization
fig, axes = plt.subplots(2, 3, figsize=(6, 2))

# Trace plots for selected ability parameters
for i, idx in enumerate([0, 49, 99]):
    axes[0, i].plot(theta_samples[:, idx], alpha=0.7, linewidth=0.5)
    axes[0, i].axhline(theta_true_centered[idx], color='r', linestyle='--',
                       linewidth=1.5, label='True')
    axes[0, i].axhline(theta_samples[:, idx].mean(), color='g', linestyle='-',
                       linewidth=1.5, label='Post. mean')
    axes[0, i].set_xlabel('Sample')
    axes[0, i].set_ylabel(f'$\\theta_{{{idx}}}$')
    axes[0, i].set_title(f'Trace: Ability {idx}')
    if i == 0:
        axes[0, i].legend(fontsize=8)

# Posterior distributions for selected difficulty parameters
for i, idx in enumerate([0, 24, 49]):
    axes[1, i].hist(beta_samples[:, idx], bins=30, density=True, alpha=0.7)
    axes[1, i].axvline(beta_true_centered[idx], color='r', linestyle='--',
                       linewidth=2, label='True')
    axes[1, i].axvline(beta_samples[:, idx].mean(), color='g', linestyle='-',
                       linewidth=2, label='Post. mean')
    axes[1, i].set_xlabel(f'$\\beta_{{{idx}}}$')
    axes[1, i].set_ylabel('Density')
    axes[1, i].set_title(f'Posterior: Difficulty {idx}')
    if i == 0:
        axes[1, i].legend(fontsize=8)

```

```
plt.tight_layout()
plt.show()

# Posterior summary statistics
theta_post_mean = theta_samples.mean(axis=0)
theta_post_std = theta_samples.std(axis=0)
beta_post_mean = beta_samples.mean(axis=0)
beta_post_std = beta_samples.std(axis=0)

print(f"\nPosterior summary:")
print(f"  Mean posterior std for abilities: {theta_post_std.mean():.3f}")
print(f"  Mean posterior std for difficulties: {beta_post_std.mean():.3f}")
print(f"  Correlation with true abilities: {np.corrcoef(theta_true_centered,
    theta_post_mean)[0,1]:.4f}")
print(f"  Correlation with true difficulties: {np.corrcoef(beta_true_centered,
    beta_post_mean)[0,1]:.4f}")
```

The posterior standard deviations quantify our uncertainty about each parameter. Parameters with more information (e.g., items answered by many models, models who answered many questions) have smaller posterior uncertainty.

4.6 Regularization and Model Selection

4.6.1 L2 Regularization as Bayesian Prior

We have seen that MAP estimation with Gaussian priors is equivalent to L2 regularization. The regularization strength λ relates to the prior variance as $\lambda = 1/\sigma^2$.

The regularized objective is:

$$\ell_{\text{reg}}(\theta, \beta) = \ell(\theta, \beta) - \frac{\lambda_\theta}{2}\|\theta\|^2 - \frac{\lambda_\beta}{2}\|\beta\|^2$$

Regularization prevents overfitting, especially when:

- Some persons have few responses (sparse data)
- Some items have extreme difficulty (near 0% or 100% pass rates)
- The model is complex (many parameters relative to data)

4.6.2 Cross-Validation for Hyperparameter Selection

How do we choose the regularization strength? Cross-validation provides a principled answer: we hold out some data, train on the rest, and evaluate prediction performance.

```
#| label: cross-validation
#| autorun: true
#| fig-cap: "Cross-validation for selecting regularization strength."

def fit_and_evaluate(Y_train_mask, Y, lambda_param, sigma_theta=None,
    sigma_beta=None):
    """Fit model on training data, evaluate on held-out data."""
    N, M = Y.shape

    # Convert lambda to prior std
    if sigma_theta is None:
        sigma_theta = 1 / np.sqrt(lambda_param + 1e-10)
    if sigma_beta is None:
        sigma_beta = 1 / np.sqrt(lambda_param + 1e-10)

    # Fit on training data
    def objective(params):
        theta = params[:N]
        beta = params[N:]
        logits = theta[:, None] - beta[None, :]

        # Only include training observations in likelihood
        ll = (Y_train_mask * (Y * logits - np.log(1 + np.exp(np.clip(logits, -500,
    500))))).sum()
        log_prior = -0.5 * ((theta**2).sum() / sigma_theta**2 +
                            (beta**2).sum() / sigma_beta**2)
        return -(ll + log_prior)

    params0 = np.zeros(N + M)
    result = minimize(objective, params0, method='L-BFGS-B', options={'maxiter': 100})

    theta_fit = result.x[:N]
    beta_fit = result.x[N:]

    # Evaluate on held-out data
    P = sigmoid(theta_fit[:, None] - beta_fit[None, :])
    test_mask = 1 - Y_train_mask

    # Log-likelihood on test set
    ll_test = (test_mask * (Y * np.log(P + 1e-10) +
               (1 - Y) * np.log(1 - P + 1e-10))).sum()
    n_test = test_mask.sum()

    return ll_test / n_test  # Average log-likelihood

def cross_validate(Y, lambda_param, n_folds=5, seed=42):
    """K-fold cross-validation for regularization strength."""
    np.random.seed(seed)
    N, M = Y.shape
```

```
48
49      # Create random fold assignments for entries
50      fold_assignment = np.random.randint(0, n_folds, (N, M))
51
52      cv_scores = []
53      for fold in range(n_folds):
54          train_mask = (fold_assignment != fold).astype(float)
55          score = fit_and_evaluate(train_mask, Y, lambda_param)
56          cv_scores.append(score)
57
58      return np.mean(cv_scores), np.std(cv_scores)
59
60  # Grid search over regularization strengths
61  lambdas = [0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 5.0]
62  cv_means = []
63  cv_stds = []
64
65  print("Cross-validation results:")
66  for lam in lambdas:
67      mean, std = cross_validate(Y, lam)
68      cv_means.append(mean)
69      cv_stds.append(std)
70      print(f"  lambda = {lam:5.3f}: CV log-lik = {mean:.4f} +/- {std:.4f}")
71
72  # Plot
73  plt.figure()
74  plt.errorbar(lambdas, cv_means, yerr=cv_stds, fmt='o-', capsize=5, markersize=8)
75  plt.xscale('log')
76  plt.xlabel('Regularization strength ($\\lambda$)')
77  plt.ylabel('Cross-validation log-likelihood')
78  plt.title('Hyperparameter Selection via Cross-Validation')
79  plt.grid(True, alpha=0.3)
80  plt.tight_layout()
81  plt.show()
82
83  best_lambda = lambdas[np.argmax(cv_means)]
84  print(f"\nBest regularization: lambda = {best_lambda}")
```

4.7 *Discussion Questions*

1. **Identifiability and Interpretation**: In AI evaluation, should we anchor the ability scale by fixing one model (e.g., GPT-4 = 0) or by centering all models? What are the implications for interpreting ability scores over time as new models are released?

2. **Bayesian vs Frequentist**: When is Bayesian inference preferred over MLE for AI benchmark analysis? Consider scenarios with limited data, extreme scores, or the need for uncertainty quantification.

3. **From Learning to Design**: This chapter focuses on estimating parameters from a fixed dataset. Chapter 6 considers how to *design* the evaluation—choosing which items to include, how to allocate testing effort, and how to ensure robustness. What aspects of the estimation methods learned here would inform good benchmark design?
4. **Transfer of Item Parameters**: If we calibrate item difficulties on one set of models (e.g., 2023 models), can we use these parameters to evaluate 2024 models? What assumptions does this require, and when might they fail?

4.8 Bibliographic Notes

4.8.1 Maximum Likelihood Estimation

The theory of maximum likelihood for IRT models is developed comprehensively in Lord and Novick (1968) and Birnbaum (1968). The joint MLE approach and its limitations (incidental parameter problem) were first formalized in Neyman and Scott (1948); Haberman (1977) proves the inconsistency specifically for the Rasch model, and Lancaster (2000) provides a modern survey. For modern computational approaches, see Baker and Kim (2004).

4.8.2 Conditional and Marginal MLE

Conditional MLE for the Rasch model was developed by Andersen (1970), who proved consistency and derived the elementary symmetric functions needed for computation. Marginal MLE was introduced by Bock and Lieberman (1970) and popularized by Bock and Aitkin (1981) using the EM algorithm.

4.8.3 EM Algorithm

The general EM algorithm was formalized by Dempster et al. (1977). Its application to IRT is detailed in Bock and Aitkin (1981). For modern treatments, see McLachlan and Krishnan (2007).

4.8.4 Bayesian IRT

Bayesian approaches to IRT were pioneered by Swaminathan and Gifford (1982) and advanced using Gibbs sampling by Algorithm **??** . Modern references include Fox (2010) and the software documentation for Stan (Carpenter et al. 2017).

4.8.5 Beta-IRT and Continuous Responses

Standard IRT models assume binary (correct/incorrect) responses, but language models provide richer signals: token probabilities in pre-training and empirical pass rates in repeated sampling. S. Truong, Tu, et al. (2025) introduce Beta-IRT, which replaces the Bernoulli loss with a Beta loss parameterized by the IRT logistic mean $\sigma(d_j(\theta_i - z_j))$ and a precision parameter ϕ. This achieves reliable calibration with as few as 2 test takers (30–60$\times$ fewer than Binary-IRT), making it practical for scaling law studies where the number of model checkpoints far exceeds what binary IRT requires. The connection between per-problem exponential scaling and aggregate power-law scaling is formalized by Schaeffer et al. (2025), who show the power-law exponent is controlled by the left-tail shape of the success probability distribution — precisely the item difficulty distribution in IRT terms.

4.8.6 Optimization Methods

L-BFGS is described in Nocedal and Wright (2006). For deep learning optimizers applied to psychometric models, see Kingma and Ba (2015) for Adam.

4.9 Exercises

4.9.1 Theoretical Exercises

Exercise 2.1 ($\star$): Derive the gradient of the Rasch model log-likelihood with respect to θ_i. Show that it equals the sum of residuals: $\frac{\partial \ell}{\partial \theta_i} = \sum_j (Y_{ij} - P_{ij})$.

Exercise 2.2 ($\star\star$): Prove that the Hessian matrix of the Rasch log-likelihood is negative semi-definite, ensuring the log-likelihood is concave.

Exercise 2.3 ($\star\star$): Show that for the Rasch model, the Fisher information for item j at ability θ is $I_j(\theta) = P_j(1 - P_j)$, and that this is maximized when $\theta = \beta_j$.

Exercise 2.4 ($\star\star\star$): Derive the EM algorithm for the 2PL model. What additional complications arise compared to the Rasch model due to the discrimination parameters?

Exercise 2.5 ($\star\star$): Show that L2 regularization on the parameters is equivalent to MAP estimation with Gaussian priors. What is the relationship between the regularization strength λ and the prior variance σ^2?

4.9.2 Computational Exercises

Exercise 2.6 ($\star\star$): Implement conditional MLE for the Rasch model. Use the fact that the conditional likelihood depends only on item parameters and can be computed using elementary symmetric functions.

Exercise 2.7 ($\star\star\star$): Implement a Gibbs sampler for the Rasch model that alternates between: - Sampling $\theta_i \mid Y, \beta$ for each person (using slice sampling) - Sampling $\beta_j \mid Y, \theta$ for each item

Compare the posterior estimates to those from Metropolis-Hastings.

Exercise 2.8 ($\star\star\star$): Implement marginal MLE using numerical quadrature to integrate out the ability parameters. Compare convergence and parameter recovery to the EM approach from this chapter.

Exercise 2.9 ($\star\star$): Implement Beta-IRT estimation. Given a response matrix $P_{ij} \in [0, 1]$ of empirical probabilities (e.g., token probabilities or pass rates), maximize the Beta log-likelihood $\sum_{i,j} \log p(P_{ij}; \mu_{ij}, \phi)$ where $\mu_{ij} = \sigma(d_j(\theta_i - z_j))$ is the IRT-predicted mean and $\phi > 0$ is a precision parameter. (a) Derive the gradient with respect to θ_i and z_j. (b) Simulate data with $M = 50$ models and $N = 200$ items, where $P_{ij} \sim \text{Beta}(\mu_{ij}\phi, (1 - \mu_{ij})\phi)$ with $\phi = 10$. (c) Compare parameter recovery of Beta-IRT vs. binary IRT (where responses are thresholded at 0.5) as a function of M. Verify the finding of S. Truong, Tu, et al. (2025) that Beta-IRT achieves reliable calibration with far fewer models than Binary-IRT.

4.9.3 Discussion Exercises

Exercise 2.9: Compare the convergence of gradient ascent, L-BFGS, and Adam on a Rasch model estimation problem. Which converges fastest? Which is most robust to different initializations?

Exercise 2.10: Investigate the sensitivity of MLE and Bayesian estimation to model misspecification. Generate data from a 2PL model but fit a Rasch model. How do the estimated abilities compare? When does the misspecification matter most?

5 Reliability

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Decompose** a benchmark's response variance into person, item, person-by-item, and residual components, derived directly from the IRT response model.
2. **Define** reliability as the person (signal) share of that variance—the generalizability coefficient—and distinguish it from validity.
3. **Estimate** the components and the reliability they imply (ANOVA/method of moments, marginal MLE, bias-corrected plug-in), and **compute** the classical score-based coefficients (Cronbach's α, McDonald's ω, split-half, test–retest, SEM), linked by Spearman–Brown.
4. **Diagnose** the sources of variability in AI evaluation—prompt sensitivity, sampling stochasticity, rater disagreement, item sampling—and map each to the component it feeds.
5. **Apply** generalizability theory to multi-facet designs, using G- and D-studies to size items and raters to a target reliability, including LLM-as-a-judge protocols.
6. **Explain** conditional reliability (precision varies with ability) and why systematic bias, such as a judge's position bias, escapes reliability analysis.
7. **Select** a model's facet structure—deciding which terms a design can identify—and **design** evaluations that control the dominant sources of variability.
8. **Diagnose** individual items with differential item functioning, dimensionality assessment, item-fit statistics, and the multitrait-multimethod matrix, connecting each to the same variance decomposition and to the validity threats catalogued in Chapter 1.

Suggested Lecture Plan

This chapter can be covered in **4 lectures** (75–90 minutes each):

Lecture 1: From responses to variance components (§5.1–5.5)

- Leaderboard flicker and the sources of variability (15 min)
- The response model and the four-term decomposition; reliability as the generalizability coefficient (20 min)
- Three estimators of the components: plug-in and its bias, Bayesian / empirical Bayes, and method of moments (30 min)
- Hands-on: variance attribution, and why the interaction needs replication (25 min)

Lecture 2: Designing the evaluation (§5.6–5.7)

- Generalizability theory and the D-study: sizing items and raters to a target reliability (35 min)
- Model selection: which facets a design can identify; held-out likelihood and WAIC (30 min)
- Hands-on: cost-optimal D-study and held-out model selection (25 min)

Lecture 3: Reliability across model families (§5.8–5.11)

- Factor models and McDonald's ω; the α–ω gap (20 min)
- Ordinal responses, LLM-as-a-judge, and agreement coefficients (κ, weighted κ, Krippendorff's α) (25 min)
- Classical test theory: α, split-half, SEM, Spearman–Brown (15 min)
- Design principles for reliable AI evaluation (10 min)
- Hands-on: ω vs α, and the agreement coefficients (20 min)

Lecture 4: Item-level diagnostics and building trustworthy evaluations

- DIF as an item-by-group interaction; Mantel-Haenszel, logistic-regression, and IRT-based tests (25 min)
- Dimensionality assessment via parallel analysis; item-fit and contamination detection (25 min)
- The multitrait-multimethod matrix; principled item construction and synthetic data (30 min)

Notation

This chapter is built on the response-model variance decomposition $\text{Var}(Y) = V_{\text{person}} + V_{\text{item}} + V_{\text{interaction}} + V_{\text{residual}}$, with reliability $\rho = V_{\text{person}} / \text{Var}(Y)$—the **generalizability coefficient**. The same components appear as the random-effects variances $\sigma^2_p, \sigma^2_i, \sigma^2_{pi}$ (plus a rater facet σ^2_r), and collapse to classical test theory's $X = T + E$ with reliability $\rho_{XX'}$, Cronbach's α, and the standard error of measurement (SEM). Other symbols: $\rho(\theta)$ and test information $I(\theta)$ for conditional reliability, and κ (Cohen's kappa) for rater agreement. The item-level diagnostics of Section 5.11 add their own symbols—α_{MH} (the Mantel-Haenszel odds ratio), dimensionality eigenvalues λ_k, and item-fit mean squares—each introduced where the tool is developed. See the notation appendix[1] for the complete reference.

[1] notation.qmd

Video Overview

A visual tour of the key concepts in this chapter — from why leaderboards flicker and the variance decomposition to estimating reliability, designing with G- and D-studies, LLM-judge agreement, and the classical coefficients.

../animations/ch5/chapter5_narrated.mp4

5.1 Sources of Variability in AI Measurement

Imagine ten language models evaluated on a reasoning benchmark, with the rankings reported to a team. The next day, a colleague re-runs the same evaluation and gets a different ranking. The top two models swap places. A model that was fourth is now second. Nothing about the models changed—only the measurement procedure differed in small ways: a slightly different prompt template, a different random seed for sampling, a different subset of annotators scoring the open-ended responses.

This is the reliability problem. Before we can ask whether a benchmark measures what it claims to measure (the validity question, addressed in Chapter 1), we must first ask a more basic question: *does the evaluation give the same answer when applied to the same thing twice?*

Reliability is not accuracy. A thermometer that consistently reads two degrees too high is reliable—it gives the same answer every time—even though it is not accurate. Conversely, a thermometer that fluctuates randomly between readings is unreliable, and therefore cannot be accurate in any useful sense. In measurement science, reliability refers to the *consistency* or *reproducibility* of a measurement procedure.

> **! Reliability is Necessary for Validity**
>
> If an evaluation procedure produces different results every time it is applied to the same model, it cannot be measuring anything about the model. Unreliable measurements are noise, regardless of how carefully the benchmark was designed. Chapter 1 addresses validity—whether we are measuring the *right* thing. This chapter addresses the prior question: are we measuring *anything* at all?

The sources of inconsistency in AI evaluation are diverse: prompt formatting, temperature sampling, annotator disagreement, benchmark item selection, scoring rubric ambiguity, and even API version drift. Each source contributes variability that can distort model rankings. This chapter provides the theoretical tools to quantify, decompose, and control these sources of variability, drawing on Classical Test Theory and Generalizability Theory from the measurement sciences, applied throughout to the specific challenges of evaluating AI systems.

These swings have identifiable causes, and which one dominates decides the cure—adding items does nothing if the problem is rater disagreement. Four mechanisms account for most of the variability in AI evaluation.

Prompt sensitivity. Even at temperature 0, a model gives different responses under different prompt formats: "Let's think step by step," a multiple-choice versus open-ended answer format, or a reordering of the few-shot examples can shift rankings (Mizrahi et al. 2024). This is the *test-format effect* of educational testing—the same knowledge assessed differently by different instruments. If model A beats B on one template but not another, their standing is prompt-dependent, not ability-dependent.

Sampling stochasticity. At temperature > 0 the same model gives different outputs to the same prompt on each run—the closest analogue to classical test–retest error. For binary items this is usually minimal (most models answer deterministically), but open-ended generations (summaries, code, essays) can earn different scores across runs. Replicated trials make it possible to measure this run-to-run noise and average it down.

Annotator and judge disagreement. For tasks that cannot be scored automatically—creative writing, safety, open-ended reasoning—human annotators or LLM judges assess quality, and they disagree. This is a distinct source that neither item difficulty nor model ability captures, so it needs its own treatment, which generalizability theory provides (Section 5.6); the special case of LLM judges—quantifying their agreement (Cohen's κ) and detecting their systematic biases—is the subject of Section 5.9.

Benchmark sampling. The benchmark is itself a sample from a larger domain: a 200-item coding benchmark is one draw from the space of all coding problems, and another draw would rank models somewhat differently. Longer tests sample the domain more thoroughly and are

more reliable—the Spearman–Brown effect (Section 5.10)—and the D-optimal item selection of Chapter 6 minimizes this variability by choosing a maximally informative sample.

To see these mechanisms bite, simulate many evaluation runs that perturb item difficulty (a prompt change), model ability (stochasticity), and the scores themselves (rater disagreement), then slice the resulting score variance by knob.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: nd-decompose
#| autorun: true
#| fig-cap: "Score variance split into signal (stable between-model differences) and
 ↪  the variability contributed by each mechanism, estimated by knocking out one knob
 ↪  at a time. Rater disagreement dominates the variability; the per-item prompt
 ↪  perturbations average out over the 50 items, and single-trial stochasticity is
 ↪  small."
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return np.where(x >= 0, 1 / (1 + np.exp(-x)), np.exp(x) / (1 + np.exp(x)))

np.random.seed(42)
N, M = 20, 50                               # models, items
theta = np.linspace(-2, 2, N)              # true abilities
beta  = np.random.normal(0, 1.5, M)        # item difficulties
sigma_prompt, sigma_rater, sigma_stoch = 0.3, 0.5, 0.2   # the three variability knobs

def run_evaluation(sp, sr, ss, seed):
    np.random.seed(seed)
```

```
    beta_eff  = beta  + np.random.normal(0, sp, M)        # prompt shifts item difficulty
    theta_eff = theta + np.random.normal(0, ss, N)        # stochasticity shifts ability
    Y = (np.random.rand(N, M) < sigmoid(theta_eff[:, None] - beta_eff[None, :])).astype(float)
    return Y.mean(1) + np.random.normal(0, sr / np.sqrt(M), N)   # + rater variability on scores

all_scores = np.array([run_evaluation(sigma_prompt, sigma_rater, sigma_stoch, r) for r in range(200)])
signal = all_scores.mean(0).var()          # between-model (stable) variance
within_var = all_scores.var(0).mean()      # within-model (run-to-run) variance

def avg_within(sp, sr, ss):                 # average run-to-run variance for a knob setting
    reps = np.array([run_evaluation(sp, sr, ss, 1000 + r) for r in range(100)])
    return reps.var(0).mean()
full = avg_within(sigma_prompt, sigma_rater, sigma_stoch)
prompt_c = max(0, full - avg_within(0, sigma_rater, sigma_stoch))   # knock out one knob at a time
rater_c  = max(0, full - avg_within(sigma_prompt, 0, sigma_stoch))
stoch_c  = max(0, full - avg_within(sigma_prompt, sigma_rater, 0))

plt.figure(figsize=(3.4, 2))
plt.bar(['model\n(signal)', 'prompt', 'rater', 'stochastic'],
        [signal, prompt_c, rater_c, stoch_c],
        color=['#2ca02c', '#ff7f0e', '#d62728', '#9467bd'], alpha=0.85)
plt.ylabel('variance'); plt.title('Variance by mechanism')
plt.tick_params(labelsize=6); plt.grid(True, alpha=0.3, axis='y'); plt.tight_layout(); plt.show()
print(f"reliability = signal / total = {signal / (signal + within_var):.3f}")
```

Most of that variability falls on models with similar abilities, so their ranks reshuffle from run to run—the flicker this section opened with, reproduced.

```
#| label: nd-ranking
#| autorun: true
#| fig-cap: "Model rank across three independent evaluation runs. Lines crossing show pairs whose ordering is not stable; well-separated models keep their place."
runs = [run_evaluation(sigma_prompt, sigma_rater, sigma_stoch, s) for s in (10, 20, 30)]
rankings = [np.argsort(-r) for r in runs]          # best model first

plt.figure(figsize=(3.4, 2))
for i in range(N):
    positions = [int(np.where(rk == i)[0][0]) for rk in rankings]
    plt.plot([1, 2, 3], positions, '-o', color=plt.cm.viridis(theta[i] / 4 + 0.5),
```

```
            alpha=0.5, ms=3, lw=0.8)
plt.gca().invert_yaxis()
plt.xticks([1, 2, 3]); plt.xlabel('evaluation run'); plt.ylabel('rank')
plt.title('Ranking instability'); plt.grid(True, alpha=0.3); plt.tight_layout();
 ↪  plt.show()
```

Equivalently, the 95% confidence interval around each model's mean score overlaps its neighbors'—marking exactly the comparisons a leaderboard cannot support.

```
#| label: nd-intervals
#| autorun: true
#| fig-cap: "Mean score per model with a 95% confidence interval over evaluation runs.
 ↪  Overlapping intervals identify ranking differences that fall within the
 ↪  variability."
means, sems = all_scores.mean(0), all_scores.std(0)
order = np.argsort(means)

plt.figure(figsize=(3.4, 2))
plt.barh(range(N), means[order], xerr=1.96 * sems[order],
         color='steelblue', alpha=0.7, capsize=2, height=0.7)
plt.yticks([]); plt.xlabel('mean score'); plt.ylabel('model (ranked)')
plt.title('Scores with 95% CI'); plt.grid(True, alpha=0.3, axis='x')
plt.tight_layout(); plt.show()
```

Rater disagreement is the dominant *fixable* source here, while prompt and item perturbations largely cancel across models on a fixed, long benchmark and stochasticity is the run-to-run noise that replication would tame. Which mechanism dominates—and how to spend a budget of items and raters against it—is what the rest of the chapter quantifies: first by decomposing the variance (Section 5.2) and estimating its parts, then by turning the decomposition into a design tool (Section 5.6).

5.2 From the Response Model to Variance Components

In Chapter 3 we modeled a single response as a sigmoid of a person term and an item term, $P_{ij} = \sigma(\theta_i - \beta_j)$. Based on this model, in addition to predictive measurement, we can ask another question: when a benchmark score varies, where does the variance come from? The answer turns out to be the entire vocabulary of this chapter—person variance, item variance, their interaction, and an irreducible residual. When we score a model on a benchmark, we rarely care about its performance on *these* particular items, graded by *this* judge, on *this* day; we care about how it would do across the whole universe of items, judges, and occasions we might have used. The observed test is only a sample from that universe, and a score is dependable only insofar as it generalizes from the sample back to the universe. Each source of variation is a way the sample can misrepresent the universe, so attributing variance to these sources is how we quantify the threat to generalization. The generalizability coefficient is the share of

score variance carried by the thing we want to generalize about—genuine differences between subjects—rather than by the luck of which items or judges we happened to draw.

Following the Rasch model of Chapter 3, let $Y_{ij} \in \{0, 1\}$ be the response of person $i \in \{1, \dots, N\}$ on item $j \in \{1, \dots, M\}$, with ability θ_i and difficulty β_j. We add one a person-by-item interaction γ_{ij} to the model. The interaction γ_{ij} lets a particular person be unusually good (or bad) on a particular item, over and above what overall ability and difficulty predict: a model that has memorized one specific question, or a question that is idiosyncratically easy for one model family, lives in γ_{ij}. We then write the log-odds of a correct response as function of the two main effects and an interaction:

$$\operatorname{logit} P(Y_{ij} = 1) = \theta_i - \beta_j + \gamma_{ij} \iff P_{ij} = P(Y_{ij} = 1 \mid \theta_i, \beta_j, \gamma_{ij}) = \sigma(\theta_i - \beta_j + \gamma_{ij}). \tag{5.1}$$

Subjects and items are not fixed but drawn from populations:

$$\theta_i \sim \mathcal{N}(0, \sigma_p^2), \qquad \beta_j \sim \mathcal{N}(0, \sigma_i^2), \qquad \gamma_{ij} \sim \mathcal{N}(0, \sigma_{pi}^2), \tag{5.2}$$

and treat the population variances—σ_p^2 across subjects, σ_i^2 across items, and σ_{pi}^2 for the person-by-item term—as the quantities to attribute, and ultimately to estimate. On the log-odds scale, the attribution is immediate, because independent effects add:

$$\operatorname{Var}(\theta_I - \beta_J + \gamma_{IJ}) = \sigma_p^2 + \sigma_i^2 + \sigma_{pi}^2. \tag{5.3}$$

The person share of this latent variance is the **reliability**, or one-facet **generalizability coefficient**,

$$\rho_\eta = \frac{\sigma_p^2}{\sigma_p^2 + \sigma_i^2 + \sigma_{pi}^2}, \tag{5.4}$$

the fraction of variation that reflects genuine differences between subjects rather than item difficulty or idiosyncratic person–item effects. This is the *idealized* reliability—what we could achieve if we observed the log-odds η directly. A real benchmark gives only binary responses, whose Bernoulli sampling adds a further, irreducible error; carrying the decomposition through the sigmoid onto that observed scale, and watching the error shrink as items accumulate, is the subject of Section 5.5.

The decomposition is not limited to two facets. An evaluation that also varies *raters*—and, across runs, *occasions*—adds their main effects and interactions; because independent effects still add on the log-odds scale, each facet brings its own variance components. For models (p) answering items (i) scored by raters (r),

$$\text{Var}(\eta) = \sigma_p^2 + \sigma_i^2 + \sigma_r^2 + \sigma_{pi}^2 + \sigma_{pr}^2 + \sigma_{ir}^2 + \sigma_{pir}^2, \tag{5.5}$$

one component per facet (σ_p^2 model, σ_i^2 item, σ_r^2 rater) and one per interaction; the first three terms are Equation 5.3, the rest are what the rater facet adds. Every mechanism of Section 5.1 is one of these components—rater disagreement is σ_r^2 and σ_{pr}^2, run-to-run drift adds an occasion facet—and generalizability theory (Section 5.6) estimates them and spends a fixed budget of items and raters to drive the error down.

One caveat attaches to the rater facet. When a rater returns a categorical verdict—a label, or which of two responses is better—rather than a number, its facet is conventionally summarized not through σ_r^2 directly but through a *chance-corrected agreement coefficient*, Cohen's κ, which plays the role of a generalizability coefficient for raters. We develop it where it is used, with LLM judges (Section 5.9).

With the estimand defined, the next three sections estimate it—plug-in from fixed effects (Section 5.3), Bayesian (Section 5.4), and method of moments (Section 5.5), in increasing distance from the response model we started with. Generalizability theory (Section 5.6) then turns the decomposition into a design tool for reliable AI evaluation.

5.3 Variance Estimation via Plug-in Method

We now have the estimand—the components $\sigma_p^2, \sigma_i^2, \sigma_{pi}^2$ of Equation 5.2 and the reliability they define—but a finite benchmark only lets us *estimate* them. Three methods do so, and we take them in order of distance from the foundations of the earlier chapters. The most direct simply reuses the IRT machinery of Chapter 3 and Chapter 4: fit each model's ability as a **fixed effect**, then ask how spread out the fitted abilities are.

Fit the Rasch abilities $\hat{\theta}_1, \ldots, \hat{\theta}_N$ by maximum likelihood with the item difficulties held fixed, and take their empirical variance as an estimate of σ_p^2. Without correction, this is a biased estimation. Write each fitted ability as the truth plus estimation error, $\hat{\theta}_i = \theta_i + e_i$, with $\theta_i \sim \mathcal{N}(0, \sigma_p^2)$ the quantity of interest and e_i the Rasch fit's error—mean zero, variance $\text{Var}(e_i) = \text{SE}(\hat{\theta}_i)^2$. The fit's error depends on the responses, not on the value of θ_i, so the two are independent and their variances add:

$$\text{Var}(\hat{\theta}_i) = \text{Var}(\theta_i) + \text{Var}(e_i) = \sigma_p^2 + \text{SE}(\hat{\theta}_i)^2.$$

The plug-in estimator is the empirical mean of these per-model variances, so (up to the usual $\frac{N-1}{N}$ finite-sample factor) it targets their average, not σ_p^2 alone:

$$\mathbb{E}\Big[\tfrac{1}{N}\textstyle\sum_i(\hat{\theta}_i - \bar{\hat{\theta}})^2\Big] \;\approx\; \sigma_p^2 + \overline{\text{SE}^2}, \qquad \overline{\text{SE}^2} = \tfrac{1}{N}\textstyle\sum_i \text{SE}(\hat{\theta}_i)^2.$$

The inflation $\overline{\mathrm{SE}^2}$ is the average sampling variance of the per-model estimates: noisy abilities spread out more than the true ones, so estimation error masquerades as between-model variance—the same confounding of signal and noise that makes the interaction and residual inseparable without replicated trials (Section 5.5). Subtracting it gives the bias-corrected estimate,

$$\hat{\sigma}_p^2 = \tfrac{1}{N}\textstyle\sum_i (\hat{\theta}_i - \bar{\hat{\theta}})^2 - \overline{\mathrm{SE}^2},$$

which is exactly the method-of-moments estimator of Section 5.5 seen on the logit scale. Each standard error is read from the Fisher information of the Rasch fit,

$$\mathrm{SE}(\hat{\theta}_i) = \Big(\textstyle\sum_{j=1}^{M} \hat{P}_{ij}\,(1-\hat{P}_{ij})\Big)^{-1/2} = I(\hat{\theta}_i)^{-1/2}, \qquad \hat{P}_{ij} = \sigma(\hat{\theta}_i - \beta_j),$$

the inverse square root of the test information $I(\theta)$ of Section 6.2.1—each item contributing $\hat{P}_{ij}(1-\hat{P}_{ij})$, most when it sits near the model's ability. The bias therefore shrinks like $1/M$: information accumulates with items, so $\overline{\mathrm{SE}^2} \sim 1/M$, and on long benchmarks the naive and corrected estimates of σ_p^2 nearly agree—but for σ_{pi}^2 they never do without replication.

The simulation plants abilities of known variance $\sigma_p^2 = 1$, draws one binary trial per cell, fits the abilities, and compares the naive and bias-corrected estimates as the benchmark lengthens.

```
#| label: plugin-estimator
#| autorun: true
#| fig-cap: "Plug-in estimation of $\\sigma_p^2$ (planted at 1.0) from one binary trial per cell. Left: at a sparse design the naive variance of the fitted abilities overshoots, while subtracting $\\overline{\\operatorname{SE}^2}$ recovers the truth. Center: the naive bias decays as items accumulate. Right: that bias is exactly the average squared standard error."
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return np.where(x >= 0, 1 / (1 + np.exp(-x)), np.exp(x) / (1 + np.exp(x)))

def fit_ability(S, beta, T, n_iter=60):
    """Per-model ML ability (item difficulties known); returns theta_hat and its SE."""
    rowp  = np.clip(S.mean(1) / T, 0.02, 0.98)
    theta = np.clip(np.log(rowp / (1 - rowp)), -4, 4)
    for _ in range(n_iter):                          # Newton steps; the Rasch log-lik is concave
        Pm   = sigmoid(theta[:, None] - beta[None, :])
        info = (T * Pm * (1 - Pm)).sum(1) + 1e-6
        theta = np.clip(theta + (S - T * Pm).sum(1) / info, -5, 5)
```

```
    Pm   = sigmoid(theta[:, None] - beta[None, :])
    info = (T * Pm * (1 - Pm)).sum(1) + 1e-9
    return theta, 1 / np.sqrt(info)                 # SE = inverse-root Fisher information

N, sd_p, sd_i, T = 150, 1.0, 1.0, 1                 # one trial per cell
true_var = sd_p ** 2

def one_run(M, seed):
    rng    = np.random.RandomState(seed)
    th, bt = rng.normal(0, sd_p, N), rng.normal(0, sd_i, M)
    S = (rng.rand(T, N, M) < sigmoid(th[:, None] - bt[None, :])[None]).sum(0).astype(float)
    th_hat, se = fit_ability(S, bt, T)
    naive = th_hat.var()
    return naive, naive - (se ** 2).mean(), (se ** 2).mean()   # naive, corrected, mean SE^2

M_grid = [40, 80, 160, 320]
R = {M: np.mean([one_run(M, 1000 + M + 11 * s) for s in range(8)], axis=0) for M in M_grid}

fig, axes = plt.subplots(1, 3, figsize=(6, 2))
naive, corr, _ = R[40]
axes[0].bar(['naive', 'bias-\ncorrected'], [naive, corr], color=['#d62728', '#1f77b4'], alpha=0.85)
axes[0].axhline(true_var, color='black', ls='--', lw=1, label='true $\\sigma_p^2$')
axes[0].set_ylabel('$\\widehat{\\sigma}_p^2$'); axes[0].set_title('Estimates ($M=40$)')
axes[0].legend(fontsize=6); axes[0].grid(True, alpha=0.3, axis='y'); axes[0].tick_params(labelsize=6)

axes[1].plot(M_grid, [R[M][0] for M in M_grid], 'o-', color='#d62728', ms=3, lw=1.4, label='naive')
axes[1].plot(M_grid, [R[M][1] for M in M_grid], 's-', color='#1f77b4', ms=3, lw=1.4, label='corrected')
axes[1].axhline(true_var, color='black', ls='--', lw=1)
axes[1].set_xscale('log'); axes[1].set_xlabel('items per model ($M$)')
axes[1].set_ylabel('$\\widehat{\\sigma}_p^2$'); axes[1].set_title('Convergence')
axes[1].legend(fontsize=6); axes[1].grid(True, alpha=0.3)

pred = [R[M][2] for M in M_grid]
emp  = [R[M][0] - true_var for M in M_grid]
lim  = max(pred) * 1.1
axes[2].plot([0, lim], [0, lim], 'k--', lw=1, alpha=0.6)
axes[2].scatter(pred, emp, c='#d62728', s=18, zorder=3)
axes[2].set_xlabel('predicted bias  $\\overline{SE^2}$'); axes[2].set_ylabel('empirical bias')
axes[2].set_title('Bias mechanism'); axes[2].grid(True, alpha=0.3)
```

```
plt.tight_layout(); plt.show()
print(f"M=40: naive={naive:.3f}, bias-corrected={corr:.3f}  (true sigma_p^2 =
 ↪  {true_var:.2f})")
```

The naive estimate overshoots by exactly the average squared standard error; subtracting it recovers σ_p^2. The correction is cheap, but the plug-in still treats each ability as a free constant, ignoring that the abilities are themselves a sample from a population—a limitation the Bayesian view of Section 5.4 removes by modelling that population directly.

The standard error of the plug-in, $\mathrm{SE}(\hat{\theta}_i) = I(\hat{\theta}_i)^{-1/2}$, depends on ability through the information $I(\theta)$: a model is pinned down precisely only where the items inform it. Reliability inherits that dependence. The single ρ of Equation 5.4 is a population average; the same response model gives a *conditional* reliability at each ability θ,

$$\rho(\theta) = 1 - \frac{1}{I(\theta)\,\sigma_p^2}, \qquad I(\theta) = \sum_{j=1}^{M} P_j(\theta)\,(1 - P_j(\theta)), \tag{5.6}$$

where σ_p^2 is the spread of abilities and $I(\theta)$ is the **test information** of Section 6.2.1. Note that $I(\theta)$ is nothing but the per-item residual $P_j(1-P_j)$—the irreducible Bernoulli noise of Equation 5.11—summed at a fixed ability: the same variance that *limits* a single response is exactly the *information* that response carries.

For AI evaluation the consequence is sharp. A benchmark can post a high overall ρ (or Cronbach's α) yet be **unreliable for the models that matter most**. Once the frontier passes the difficulty of every item, $P_j(\theta) \to 1$, the information $I(\theta) \to 0$, and $\rho(\theta)$ collapses—benchmark saturation that a single reliability number hides.

```
#| label: conditional-reliability
#| autorun: true
#| fig-cap: "Conditional reliability $\\rho(\\theta)$ for an item pool clustered at
 ↪  easy difficulties versus one spread across the range. The easy pool looks fine on
 ↪  average but its reliability collapses for high-ability (frontier) models, whose
 ↪  responses its items barely inform."
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return np.where(x >= 0, 1 / (1 + np.exp(-x)), np.exp(x) / (1 + np.exp(x)))

theta = np.linspace(-3, 3, 200)
sigma_p2 = 1.0 ** 2
M = 40
beta_easy  = np.random.RandomState(0).normal(-1.0, 0.5, M)   # items clustered at easy
 ↪  difficulties
```

```
beta_broad = np.linspace(-2.5, 2.5, M)                          # items spread across the range

def cond_reliability(betas):
    I = np.sum([sigmoid(theta - b) * (1 - sigmoid(theta - b)) for b in betas], axis=0)
    return np.clip(1 - 1 / (I * sigma_p2 + 1e-9), 0, 1)

plt.figure(figsize=(3.6, 2.2))
plt.plot(theta, cond_reliability(beta_easy),  color='#d62728', lw=1.8, label='easy items')
plt.plot(theta, cond_reliability(beta_broad), color='#2ca02c', lw=1.8, label='broad items')
plt.axhline(0.90, color='black', ls='--', lw=1, alpha=0.6)
plt.xlabel('ability $\\theta$  (frontier models at right)')
plt.ylabel('conditional reliability $\\rho(\\theta)$')
plt.title('Reliability varies with ability'); plt.legend(fontsize=7); plt.ylim(0, 1)
plt.grid(True, alpha=0.3); plt.tight_layout(); plt.show()
```

The fix is the design side of efficiency (Chapter 6): pick items whose difficulty matches the ability range of interest, or select them adaptively per model (Section 6.2.1, Section 6.2.2). Targeted items hold $\rho(\theta)$ high exactly where models must be told apart.

5.4 Variance Estimation via Bayesian Method

The random-effects model Equation 5.2 is already a Bayesian hierarchical model: the distribution $\theta_i \sim \mathcal{N}(0, \sigma_p^2)$ is a **prior** on each model's ability, and the variance components $\sigma_p^2, \sigma_i^2, \sigma_{pi}^2$ are its **hyperparameters**. Estimation then comes in two flavors, depending on what we do with those hyperparameters.

Empirical Bayes (marginal maximum likelihood). Treat the abilities as latent and integrate them out, leaving the marginal likelihood—the *evidence*—as a function of the hyperparameter, then pick the value that maximizes it. Writing the evidence at a candidate variance s^2,

$$\mathcal{L}(s^2) = \int \Big[\prod_{i,j} P_{ij}^{Y_{ij}} (1 - P_{ij})^{1 - Y_{ij}} \Big] \, p(\theta, \beta, \gamma \mid s^2) \, d\theta \, d\beta \, d\gamma, \qquad P_{ij} = \sigma(\theta_i - \beta_j + \gamma_{ij}),$$

with the population (prior) density $p(\theta, \beta, \gamma \mid s^2) = \prod_i \mathcal{N}(\theta_i; 0, s_p^2) \prod_j \mathcal{N}(\beta_j; 0, s_i^2) \prod_{ij} \mathcal{N}(\gamma_{ij}; 0, s_{pi}^2)$, the estimate is the maximizer

$$\hat{\sigma}^2 = \arg\max_{s^2} \log \mathcal{L}(s^2),$$

a search for the prior scale that best explains the data—a hyperparameter point estimate. The Bernoulli noise sits inside the likelihood, so no separate residual correction is needed and the

estimates are logit-scale. With item difficulties fixed and only abilities random, the evidence is a product of one-dimensional integrals,

$$\mathcal{L}(s^2) = \prod_{i=1}^{N} \int_{\mathbb{R}} \prod_{j=1}^{M} \sigma(\theta - \beta_j)^{Y_{ij}} \big(1 - \sigma(\theta - \beta_j)\big)^{1 - Y_{ij}} \, \mathcal{N}(\theta; 0, s^2) \, d\theta,$$

which a grid evaluates exactly.

Full Bayes. Rather than point-estimate the hyperparameter, place a hyperprior $p(s^2)$ on it and report the whole posterior,

$$p(s^2 \mid Y) \;\propto\; \mathcal{L}(s^2)\, p(s^2),$$

summarized by its mean and a 95% credible interval. Empirical Bayes is then the special case that keeps only the peak of this posterior (the evidence maximizer, under a flat hyperprior); full Bayes keeps the spread, which is the honest uncertainty about σ_p^2 from a finite sample. For the multi-facet model the posterior is high-dimensional and is sampled by MCMC (NumPyro or PyMC); for the one-facet case with difficulties fixed, the same one-dimensional grid that gave the evidence gives the posterior directly.

The simulation evaluates the evidence over candidate ability spreads s, marks the empirical-Bayes maximizer, and overlays the full posterior (evidence times a flat hyperprior, normalized) with its mean and 95% credible interval.

```
#| label: bayes-estimator
#| autorun: true
#| fig-cap: "Estimating the ability spread $\\sigma_p$ from one binary trial per cell. The curve is the posterior (evidence $\\times$ flat hyperprior, normalized); empirical Bayes / marginal MLE keeps only its peak, while full Bayes reports the mean and the shaded 95% credible interval. The truth is $\\sigma_p = 1$."
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return np.where(x >= 0, 1 / (1 + np.exp(-x)), np.exp(x) / (1 + np.exp(x)))

N, M, T, sd_p = 150, 40, 1, 1.0
rng   = np.random.RandomState(0)
theta = rng.normal(0, sd_p, N)
beta  = rng.normal(0, 1.0, M)
S = (rng.rand(T, N, M) < sigmoid(theta[:, None] - beta[None, :])[None]).sum(0).astype(float)

def log_evidence(s, K=121):                                  # log p(Y | s), abilities integrated out
    g  = np.linspace(-5, 5, K)
```

```
    P  = sigmoid(g[:, None] - beta[None, :])
    LL = S @ np.log(P + 1e-12).T + (T - S) @ np.log(1 - P + 1e-12).T
    a  = LL + (-0.5 * (g / s) ** 2 - np.log(s))[None, :] + np.log(g[1] - g[0])
    m  = a.max(1, keepdims=True)
    return (m[:, 0] + np.log(np.exp(a - m).sum(1))).sum()

s_grid = np.linspace(0.3, 2.0, 60)
ds     = s_grid[1] - s_grid[0]
logE   = np.array([log_evidence(s) for s in s_grid])

s_mml  = s_grid[logE.argmax()]                          # empirical Bayes: evidence peak
post   = np.exp(logE - logE.max()); post /= post.sum() * ds   # full Bayes: flat hyperprior, normalized
cdf    = np.cumsum(post) * ds; cdf /= cdf[-1]
s_mean = (s_grid * post).sum() * ds
lo, hi = s_grid[np.searchsorted(cdf, 0.025)], s_grid[np.searchsorted(cdf, 0.975)]

plt.figure(figsize=(3.8, 2.3))
plt.plot(s_grid, post, color='#1f77b4', lw=1.8, label='posterior $p(\\sigma_p \\mid Y)$')
plt.fill_between(s_grid, post, where=(s_grid >= lo) & (s_grid <= hi), color='#1f77b4', alpha=0.2, label='95% CI')
plt.axvline(s_mml, color='#d62728', lw=1.4, label=f'EB / MML peak = {s_mml:.2f}')
plt.axvline(s_mean, color='#1f77b4', ls=':', lw=1.4, label=f'posterior mean = {s_mean:.2f}')
plt.axvline(1.0, color='black', ls='--', lw=1, label='true = 1.0')
plt.xlabel('ability spread $\\sigma_p$'); plt.ylabel('density'); plt.yticks([])
plt.title('Empirical Bayes vs full Bayes'); plt.legend(fontsize=6)
plt.tight_layout(); plt.show()
print(f"empirical Bayes (MML) sigma_p = {s_mml:.3f};  full Bayes mean = {s_mean:.3f}, 95% CI = [{lo:.2f}, {hi:.2f}]")
```

The two views answer different questions: empirical Bayes returns the single most-supported σ_p, while full Bayes returns the range the data leave open—wide here, because one binary response per cell is weak evidence. Both, like the plug-in, work on the latent scale. The last method drops the model entirely and reads the components off the observed scores.

5.5 *Variance Estimation via Method of Moments*

The plug-in and Bayesian methods fit the response model; the third estimator fits nothing, reading the components off the observed scores by method of moments. Because the scores are binary, the components it exposes live on the probability scale, which we develop first. Section 5.2 worked on the log-odds scale, where the three components add cleanly. A benchmark, though, produces only binary responses $Y \sim \text{Bernoulli}(P_{IJ})$—not the latent predictor. We now carry the decomposition through the sigmoid to $\text{Var}(Y)$: the nonlinear link recasts the three terms onto the **probability scale** P_{ij}, where the decomposition stays exact and orthogonal,

and adds a fourth—the Bernoulli noise. We work on the probability scale because it yields that residual for free. (The orthogonal decomposition that follows needs only that persons and items are sampled independently, not that they are Gaussian.) The recipe is **analysis of variance (ANOVA)**: split the total variance into additive, non-overlapping contributions, one per source—here persons, items, their interaction, and noise—by measuring how far each person's and each item's average sits from the overall average. Carried out on the cell-probability function P_{ij} rather than on raw scores, the version we use is the *functional ANOVA*, or Hoeffding–Sobol, decomposition.

We introduce some notation

$$\bar{P} = \mathbb{E}_{I,J}[P_{IJ}], \qquad P_{i\bullet} = \mathbb{E}_J[P_{iJ}], \qquad P_{\bullet j} = \mathbb{E}_I[P_{Ij}], \tag{5.7}$$

and define the person main effect $P_{i\bullet} - \bar{P}$, the item main effect $P_{\bullet j} - \bar{P}$, and the interaction $P_{ij} - P_{i\bullet} - P_{\bullet j} + \bar{P}$, so that

$$P_{ij} = \bar{P} + \underbrace{(P_{i\bullet} - \bar{P})}_{\text{person}} + \underbrace{(P_{\bullet j} - \bar{P})}_{\text{item}} + \underbrace{(P_{ij} - P_{i\bullet} - P_{\bullet j} + \bar{P})}_{\text{interaction}}. \tag{5.8}$$

Each term removes progressively more conditioning: the person effect is how much person i shifts the success probability after averaging over all items, the item effect is the analogous shift for item j, and the interaction is whatever remains—the part of the cell that neither margin explains. By construction these effects are mean-zero and mutually **orthogonal** (each averages to zero over any index it retains), so every cross-covariance vanishes and the variance of the success probability splits with no cross terms:

$$\operatorname{Var}(P_{IJ}) = \underbrace{\operatorname{Var}_I(P_{I\bullet})}_{V_{\text{person}}} + \underbrace{\operatorname{Var}_J(P_{\bullet J})}_{V_{\text{item}}} + \underbrace{\mathbb{E}\big[(P_{IJ} - P_{I\bullet} - P_{\bullet J} + \bar{P})^2\big]}_{V_{\text{interaction}}}. \tag{5.9}$$

The three terms above account for variance in the probability of a correct response, but the graded outcome Y is a Bernoulli draw from that probability. The law of total variance, conditioning on the sampled cell (I, J), supplies the last piece:

$$\operatorname{Var}(Y) = \underbrace{\operatorname{Var}\big(\mathbb{E}[Y \mid I, J]\big)}_{\operatorname{Var}(P_{IJ}) = V_{\text{person}} + V_{\text{item}} + V_{\text{interaction}}} + \underbrace{\mathbb{E}\big[\operatorname{Var}(Y \mid I, J)\big]}_{V_{\text{residual}}}. \tag{5.10}$$

Since $\operatorname{Var}(Y \mid I, J) = P_{IJ}(1 - P_{IJ})$, the residual is the average Bernoulli noise,

$$V_{\text{residual}} = \mathbb{E}\big[P_{IJ}\,(1 - P_{IJ})\big], \tag{5.11}$$

the irreducible measurement noise of a single graded trial—largest near $P_{ij} = 0.5$, vanishing as items become trivially easy or impossibly hard. Collecting the four pieces gives the decomposition that organizes the rest of this chapter:

$$\text{Var}(Y) = V_{\text{person}} + V_{\text{item}} + V_{\text{interaction}} + V_{\text{residual}} \tag{5.12}$$

Dividing through by $\text{Var}(Y)$ turns each term into a share of total variance. The person share is the **observed-scale reliability**—the analogue of the idealized ρ_η (Equation 5.4), now with the Bernoulli residual in the denominator, which a single graded response cannot escape:

$$\rho = \frac{V_{\text{person}}}{V_{\text{person}} + V_{\text{item}} + V_{\text{interaction}} + V_{\text{residual}}}. \tag{5.13}$$

Equation 5.12 is the variance-component decomposition of a one-facet Generalizability study, derived here from the response model rather than postulated. Each probability-scale term names a classical G-theory component—the score-scale image of a latent variance from Equation 5.2:

Term in Equation 5.12	G-theory component	What it measures
V_{person}	universe-score (person) variance	real differences between test-takers (the signal)
V_{item}	item variance	some items are just harder than others
$V_{\text{interaction}}$	person×item variance	a person is unusually strong/weak on specific items
V_{residual}	residual	irreducible Bernoulli grading noise
ρ (Equation 5.13)	generalizability coefficient / ICC	reliability

The same logic extends to more facets—raters, occasions—each adding its own variance components, the multi-facet decomposition of Equation 5.5; generalizability theory (Section 5.6) estimates those components and designs around them.

! WHY REPLICATED TRIALS ARE NEEDED

With a *single* graded response per cell, the estimated cell probability $\hat{P}_{ij}$ is just 0 or 1, and the interaction $V_{\text{interaction}}$ is perfectly confounded with the Bernoulli residual V_{residual}—there is no way to tell an idiosyncratic person-item effect apart from a lucky guess. Separating them requires **replication**: multiple trials per person-item cell, so that $\hat{P}_{ij}$ is a within-cell average. This is the statistical reason a long-form evaluation record carries a `trial` field alongside the person and item identifiers, and it is exactly the move from a one-facet to a two-facet G-study.

These observed-scale components are estimated straight from the scores by the classic G-study route—**analysis of variance / method of moments**. For a crossed person-by-item design with n_p persons, n_i items, and one trial per cell, write the person (row), item (column), and grand means of the observed responses,

$$\bar{Y}_{i\bullet} = \frac{1}{n_i}\sum_j Y_{ij}, \qquad \bar{Y}_{\bullet j} = \frac{1}{n_p}\sum_i Y_{ij}, \qquad \bar{Y}_{\bullet\bullet} = \frac{1}{n_p n_i}\sum_{i,j} Y_{ij}, \tag{5.14}$$

and form the person, item, and interaction **mean squares**

$$MS_p = \frac{n_i}{n_p - 1}\sum_i (\bar{Y}_{i\bullet} - \bar{Y}_{\bullet\bullet})^2, \quad MS_i = \frac{n_p}{n_i - 1}\sum_j (\bar{Y}_{\bullet j} - \bar{Y}_{\bullet\bullet})^2, \quad MS_{pi} = \frac{\sum_{i,j}(Y_{ij} - \bar{Y}_{i\bullet} - \bar{Y}_{\bullet j} + \bar{Y}_{\bullet\bullet})^2}{(n_p - 1)(n_i - 1)}, \tag{5.15}$$

the average squared person, item, and residual-cell deviations, scaled by their degrees of freedom. Treating persons and items as randomly sampled (the only assumption behind Equation 5.9), their expectations are

$$\mathbb{E}[MS_p] = (V_{\text{interaction}} + V_{\text{residual}}) + n_i\, V_{\text{person}}, \quad \mathbb{E}[MS_i] = (V_{\text{interaction}} + V_{\text{residual}}) + n_p\, V_{\text{item}}, \quad \mathbb{E}[MS_{pi}] = V_{\text{interaction}} + \tag{5.16}$$

so equating observed to expected and solving gives $\hat{V}_{\text{person}} = (MS_p - MS_{pi})/n_i$ and $\hat{V}_{\text{item}} = (MS_i - MS_{pi})/n_p$, while MS_{pi} estimates the interaction and residual together—fused until trials are replicated (Section 5.6). This last step is what names the method: a *moment* is a summary statistic whose expected value the model predicts—a mean, a variance, or here a mean square—and the **method of moments** estimates parameters by setting each observed moment equal to its predicted expectation and solving the resulting equations. Unlike the plug-in and Bayesian routes it maximizes no likelihood and assumes no prior, which makes it the cheapest of the three but also the bluntest—it uses only these low-order summaries of the data. The empirical cell proportions $\hat{P}_{ij}$ give the same components through the functional ANOVA of Equation 5.9.

The simulation below builds the response model, reads off the four population components, and then asks two questions: can we recover them from finite data, and how does the interaction erode reliability? We take it in three short steps.

First, construct the cell-probability matrix from the model and compute the four components directly from it.

```
#| label: va-decompose
#| autorun: true
```

```
#| fig-cap: "The four variance components of @eq-full-decomp as shares of total response variance, computed from the true cell probabilities. For a single binary trial the person share---the signal---is small next to item difficulty and Bernoulli noise."
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return np.where(x >= 0, 1 / (1 + np.exp(-x)), np.exp(x) / (1 + np.exp(x)))

np.random.seed(42)
N, M = 60, 40                                   # persons (models), items
gamma_scale = 0.6                               # person-by-item interaction scale

theta = np.random.normal(0, 1.0, N)             # person abilities
beta  = np.random.normal(0, 1.5, M)             # item difficulties
gamma = np.random.normal(0, gamma_scale, (N, M))  # person x item interaction
P     = sigmoid(theta[:, None] - beta[None, :] + gamma)   # cell probabilities

def anova_components(Pmat):
    """Functional-ANOVA components of a cell-probability matrix: person, item, interaction."""
    g   = Pmat.mean()
    row = Pmat.mean(1) - g                      # person effect   (P_i. - Pbar)
    col = Pmat.mean(0) - g                      # item effect     (P_.j - Pbar)
    inter = Pmat - row[:, None] - col[None, :] - g
    return (row ** 2).mean(), (col ** 2).mean(), (inter ** 2).mean()

V_person, V_item, V_inter = anova_components(P)
V_resid = (P * (1 - P)).mean()                  # Bernoulli residual
shares  = np.array([V_person, V_item, V_inter, V_resid])
shares /= shares.sum()

labels = ['Person\n(signal)', 'Item', 'Person×\nItem', 'Residual']
plt.figure(figsize=(3.2, 2))
plt.bar(labels, shares, color=['#2ca02c', '#1f77b4', '#ff7f0e', '#8c564b'])
plt.ylabel('share of Var(Y)'); plt.title('Variance decomposition')
plt.tick_params(labelsize=6); plt.tight_layout(); plt.show()
print(f"shares: person={shares[0]:.3f}  item={shares[1]:.3f}  "
      f"interaction={shares[2]:.3f}  residual={shares[3]:.3f}")
```

Estimating the components from finite data recovers the main effects, but with a single trial per cell the interaction is inseparable from Bernoulli noise: its estimate is badly inflated, and only converges to the truth once each cell is replicated.

```
#| label: va-identifiability
#| autorun: true
```

```
#| fig-cap: "Estimated person-by-item interaction variance against the number of trials per cell. With one trial it absorbs the Bernoulli residual and overshoots; replication separates the two, converging to the true value (dashed)."
def estimate_interaction(T, seed):
    """Estimate the interaction component from T binary trials per cell."""
    rng  = np.random.RandomState(seed)
    Phat = (rng.rand(T, N, M) < P[None]).mean(0)    # within-cell success rate
    return anova_components(Phat)[2]

trials  = [1, 2, 3, 5, 8, 12, 20, 40]
Vpi_hat = [np.mean([estimate_interaction(T, 500 + s) for s in range(15)]) for T in trials]

plt.figure(figsize=(3.2, 2))
plt.plot(trials, Vpi_hat, 'o-', color='#ff7f0e', label='estimated')
plt.axhline(V_inter, color='black', linestyle='--', linewidth=1, label='true')
plt.xlabel('trials per cell ($T$)'); plt.ylabel('interaction variance')
plt.title('Interaction needs replication'); plt.legend(fontsize=7)
plt.grid(True, alpha=0.3); plt.tight_layout(); plt.show()
print(f"V_interaction:  T=1 -> {Vpi_hat[0]:.4f}   T=40 -> {Vpi_hat[-1]:.4f}   true -> {V_inter:.4f}")
```

Finally, the interaction is what erodes reliability: as the person-by-item term grows, more of the variance is idiosyncratic, and the person share—the generalizability coefficient—falls.

```
#| label: va-reliability
#| autorun: true
#| fig-cap: "Reliability (the person share of variance) as a function of the interaction scale $\\sigma_\\gamma$. Stronger person-by-item interaction means a model's standing depends more on which items it happens to face, lowering reliability."
scales = np.linspace(0.0, 1.6, 14)
rho = []
for s in scales:
    g  = np.random.normal(0, s, (N, M))
    Ps = sigmoid(theta[:, None] - beta[None, :] + g)
    vp, vi, vpi = anova_components(Ps)
    rho.append(vp / (vp + vi + vpi + (Ps * (1 - Ps)).mean()))

plt.figure(figsize=(3.2, 2))
plt.plot(scales, rho, '-', color='#2ca02c', linewidth=1.8)
plt.xlabel('interaction scale $\\sigma_\\gamma$'); plt.ylabel('reliability $\\rho$')
plt.title('Reliability vs interaction'); plt.grid(True, alpha=0.3)
plt.ylim(0, max(rho) * 1.1); plt.tight_layout(); plt.show()
```

These estimates use the full response matrix; when only total scores are available, the classical test theory coefficients of Section 5.10 recover the same reliability directly.

5.6 Design for Generalizability

Equation 5.5 wrote the variance of a multi-facet design as a sum of components. A *G-study* estimates those components from data—by the method-of-moments or REML routes of Section 5.5 and Section 5.4. Generalizability theory's distinctive step is the D-study (Cronbach et al. 1972): given the components, design the evaluation—how many items and raters buy a target reliability. That, and the design principles it implies, close the chapter.

The generalizability coefficient is the ratio of true score variance to true-plus-relative-error variance:

$$G = \frac{\sigma_p^2}{\sigma_p^2 + \sigma_{\text{rel}}^2}$$

where the relative error variance for a design with n_i items and n_r raters is:

$$\sigma_{\text{rel}}^2 = \frac{\sigma_{pi}^2}{n_i} + \frac{\sigma_{pr}^2}{n_r} + \frac{\sigma_{pir,e}^2}{n_i \cdot n_r}$$

The D-study plugs the estimated components into this coefficient for a *planned* design and asks how many items and raters it takes to reach, say, $G \geq 0.90$. The key insight is that facets can be traded off against each other. If rater variance dominates (σ_{pr}^2 is large), increasing n_r is more cost-effective than increasing n_i. If item sampling variance dominates (σ_{pi}^2 is large), more items help more than more raters. The D-study makes these tradeoffs explicit and quantitative. Take the seven variance components of a simulated model×item×rater design as given (a G-study estimates them; Section 5.5, Section 5.4) and read off the breakdown.

```
#| label: gd-components
#| autorun: true
#| fig-cap: "The seven variance components of a crossed model×item×rater design. The model component is the signal; everything else is a source of error a D-study can design against."
import numpy as np
import matplotlib.pyplot as plt

# Seven variance components of a crossed model x item x rater design.
# A G-study estimates these from data; we take them as given to focus on design.
true_var = {'p': 1.0, 'i': 0.5, 'r': 0.3, 'pi': 0.4, 'pr': 0.3, 'ir': 0.1, 'e': 0.2}

names  = ['model\n(signal)', 'item', 'rater', 'model×\nitem', 'model×\nrater', 'residual']
keys   = ['p', 'i', 'r', 'pi', 'pr', 'e']
colors = ['#2ca02c', '#1f77b4', '#d62728', '#ff7f0e', '#9467bd', '#8c564b']

plt.figure(figsize=(3.6, 2))
```

```
plt.bar(names, [true_var[k] for k in keys], color=colors, alpha=0.85)
plt.ylabel('variance'); plt.title('Variance components (G-study)')
plt.xticks(rotation=30, ha='right'); plt.tick_params(labelsize=6)
plt.grid(True, alpha=0.3, axis='y'); plt.tight_layout(); plt.show()
```

The D-study plugs those components into the generalizability coefficient for a *planned* design of n_i items and n_r raters. Sweeping both traces the design frontier—and shows that items and raters substitute for each other only up to a point.

```
#| label: gd-dstudy
#| autorun: true
#| fig-cap: "Generalizability coefficient as a function of the number of items and
 ↪  raters in a planned design. Black contours mark $G = 0.80$ and $0.90$; the
 ↪  curvature shows that beyond a point, adding the cheaper facet stops helping."
n_items  = np.arange(5, 51, 5)
n_raters = np.arange(1, 11)

def G_coeff(ni, nr):
    rel = true_var['pi'] / ni + true_var['pr'] / nr + true_var['e'] / (ni * nr)
    return true_var['p'] / (true_var['p'] + rel)

G = np.array([[G_coeff(ni, nr) for ni in n_items] for nr in n_raters])

plt.figure(figsize=(3.6, 2))
im = plt.contourf(n_items, n_raters, G, levels=np.arange(0.5, 1.01, 0.05),
 ↪  cmap='RdYlGn')
plt.contour(n_items, n_raters, G, levels=[0.80, 0.90], colors='black', linewidths=1.2)
plt.xlabel('number of items'); plt.ylabel('number of raters')
plt.title('G-coefficient (D-study)'); plt.colorbar(im, shrink=0.85)
plt.tight_layout(); plt.show()
```

Add costs—items are cheap (an API call), raters expensive (annotation)—and the D-study becomes a constrained optimization: for each budget, find the (n_i, n_r) that maximizes G.

```
#| label: gd-cost
#| autorun: true
#| fig-cap: "Best achievable generalizability coefficient as a function of total
 ↪  budget, optimizing the split between items and raters at each level. The dashed
 ↪  line marks the $G = 0.90$ target."
cost_item, cost_rater = 1.0, 5.0
budgets = np.arange(50, 501, 10)

best_G = []
for B in budgets:
    g = 0.0
    for ni in range(5, int(B / cost_item) + 1):
        nr = max(1, int((B - ni * cost_item) / (ni * cost_rater)))
```

```
        if ni * cost_item + ni * nr * cost_rater <= B:
            g = max(g, G_coeff(ni, nr))
    best_G.append(g)

plt.figure(figsize=(3.6, 2))
plt.plot(budgets, best_G, color='steelblue', lw=1.5)
plt.axhline(0.90, color='red', ls='--', lw=1, label='G = 0.90')
plt.xlabel('budget'); plt.ylabel('optimal G-coefficient')
plt.title('Cost-optimal design'); plt.legend(fontsize=7)
plt.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

for B, g in zip(budgets, best_G):
    if g >= 0.90:
        print(f"minimum budget for G >= 0.90: {B}")
        break
```

5.7 *Model Selection: Which Facets to Control*

The D-study fixed *how many* items and raters to collect, but it took the **facet structure** as given—person and item main effects, a person-by-item interaction, perhaps a rater facet. Choosing that structure is the other half of design. The chapter opened with the slogan that controlling more sources of variation yields more dependable conclusions, and that is true for *bias*: a richer decomposition stops misattributing variance. But every added term costs **identifiability** and **estimation variance**, so more is not free. Deciding which terms earn their place is model selection.

Identifiability comes first. A term that cannot be separated from another cannot be selected, no matter how clever the criterion. The leading example is already in hand: with one trial per cell, the person-by-item interaction σ^2_{pi} is perfectly confounded with the Bernoulli residual (Section 5.5)—no statistic distinguishes an idiosyncratic person–item effect from an unlucky draw. So model selection and design are coupled: *the design decides which models are even askable.* To test for an interaction, the D-study must budget replicated trials per cell; otherwise the question is unanswerable and the honest move is to fuse interaction and residual into one term.

Two interactions, two held-out tests. "Do we need the interaction?" hides two very different questions. As an **iid** effect $\gamma_{ij} \sim \mathcal{N}(0, \sigma^2_{pi})$, the interaction says nothing about an unseen (model, item) cell—its posterior there is the prior—so it is testable only by holding out *replicated trials* of seen cells. As a **structured** effect—unequal item discriminations (the 2PL of Chapter 3) or extra latent dimensions—it changes predictions for unseen cells, and is tested by holding out *cells* with the masking schemes of Section 7.5.1. The second is usually the real question: "is Rasch enough, or do we need 2PL/multidimensional structure?" is prescriptive-vs-descriptive model choice in disguise.

The criterion. The natural starting point—comparing likelihood on held-out data—is correct, with three refinements. (i) Use the **marginal** predictive likelihood, integrating out the random effects, not the conditional likelihood at point estimates, which always flatters the richer model.

(ii) Hold out the right thing (cells vs. trials, per the previous paragraph). (iii) **WAIC** and **PSIS-LOO** (Watanabe 2010; Vehtari et al. 2017) approximate that held-out likelihood cheaply and handle the random-effect integration; a likelihood-ratio test of $\sigma^2_{pi} = 0$ is a *boundary* test (null on the edge of the parameter space), so its reference law is a $\frac{1}{2}\chi^2_0 + \frac{1}{2}\chi^2_1$ mixture rather than a plain χ^2_1—easy to misuse, another reason to prefer cross-validation. Pair any of these with residual diagnostics—Rasch infit/outfit, or the inter-item correlations of the item analysis (Section 3.3)—which show *where* the main-effects model misfits, not merely that it does.

The simulation generates replicated responses from an additive-plus-interaction model and compares two fits on held-out trials—a **main-effects** model (person and item effects only) and a **saturated** model (a free probability per cell, the interaction taken to the limit)—as replication grows.

```
#| label: ms-holdout
#| autorun: true
#| fig-cap: "Held-out log-likelihood of a main-effects model vs. a saturated (per-cell, full-interaction) model, against trials per cell. With no real interaction (left) the main-effects model always generalizes better---the saturated model overfits, badly at low replication. With a real interaction (right) the saturated model needs enough replicated trials to identify it before it overtakes; under scarce data the simpler, biased model still wins. Held-out likelihood selects the right structure, and what counts as 'right' depends on how much replication the design bought."
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.RandomState(0)
N, M = 50, 50
a = rng.normal(0, 0.12, N)                      # person main effect (probability scale)
b = rng.normal(0, 0.12, M)                      # item main effect

def true_P(c):                                  # additive main effects + interaction of scale c
    g = rng.normal(0, 0.12, (N, M))
    return np.clip(0.5 + a[:, None] + b[None, :] + c * g, 0.02, 0.98)

def heldout_ll(P, T):                           # T train trials, T held-out trials per cell
    tr = (rng.rand(T, N, M) < P[None]).mean(0)                 # train cell rates
    te = (rng.rand(T, N, M) < P[None]).astype(float)          # held-out trials
    pb, pi, pj = tr.mean(), tr.mean(1), tr.mean(0)
    P_add = np.clip(pi[:, None] + pj[None, :] - pb, 0.02, 0.98)   # main-effects reconstruction
    P_sat = np.clip(tr, 0.02, 0.98)                                # saturated (per-cell) model
    ll = lambda Q: (te * np.log(Q[None]) + (1 - te) * np.log(1 - Q[None])).mean()
    return ll(P_add), ll(P_sat)
```

```
Ts = [1, 2, 4, 8, 16, 32]
fig, axes = plt.subplots(1, 2, figsize=(5.4, 2), sharey=True)
for ax, (c, title) in zip(axes, [(0.0, 'no interaction'), (1.0, 'real interaction')]):
    P = true_P(c)
    res = np.array([heldout_ll(P, T) for T in Ts])
    ax.plot(Ts, res[:, 0], 'o-', color='#1f77b4', ms=3, label='main effects')
    ax.plot(Ts, res[:, 1], 's-', color='#d62728', ms=3, label='+ interaction
    ↪ (saturated)')
    ax.set_title(title, fontsize=8); ax.set_xscale('log', base=2)
    ax.set_xlabel('trials per cell'); ax.grid(True, alpha=0.3)
axes[0].set_ylabel('held-out log-lik.'); axes[0].legend(fontsize=5.5)
plt.tight_layout(); plt.show()
P0 = true_P(0.0); P1 = true_P(1.0)
a0 = heldout_ll(P0, 2); a1 = heldout_ll(P1, 32)
print(f"no interaction, T=2 : main={a0[0]:.4f}  saturated={a0[1]:.4f}  (main wins)")
print(f"real interaction, T=32: main={a1[0]:.4f}  saturated={a1[1]:.4f}  (saturated
↪ wins)")
```

What actually decides it. Held-out likelihood answers "does the term improve prediction," but the chapter's framing points at the sharper question: *does the term change the conclusion of interest*—the reliability number, the ranking? Adding σ^2_{pi} lowers bias in the reliability estimate but raises its variance, and in the small-data regime the simpler model can yield a **more dependable** conclusion even while slightly biased. A term that is statistically "significant" yet leaves the ranking untouched has not earned its place; one that drops reliability from 0.9 to 0.6 plainly has. Select the model against the decision, not against fit in the abstract—which is exactly why model selection belongs next to design rather than apart from it.

5.8 Generalizability for Factor Models

Every reliability so far has assumed items are *equally diagnostic*: in the Rasch model behind Equation 5.4 each item relates to ability with the same unit slope, differing only in difficulty. Real benchmark items violate this—a few separate strong models from weak ones sharply, while many barely discriminate at all. The **factor model** (Section 7.2) captures that heterogeneity by giving each item its own loading, and its reliability coefficient is **McDonald's omega**. Rather than quote the formula, we derive it from a generative model.

Give model i a latent ability $\theta_i \sim \mathcal{N}(0, \sigma^2_\theta)$, and let item j load on it with its own slope λ_j—the *loading*, the linear cousin of an IRT discrimination—around a baseline ν_j (the cousin of difficulty):

$$X_{ij} = \nu_j + \lambda_j \theta_i + \varepsilon_{ij}, \qquad \varepsilon_{ij} \sim \mathcal{N}(0, \psi_j), \quad \theta \perp \varepsilon. \tag{5.17}$$

This is the **one-factor (congeneric) model**: a single common factor θ generates every item, but with item-specific strength λ_j and item-specific noise variance ψ_j (the *uniqueness*). For graded

or continuous scores—rubric ratings, reward-model outputs—it holds directly; for binary items it is the linear approximation to the logistic model of Chapter 3. Equal loadings $\lambda_j \equiv \lambda$ recover the Rasch/tau-equivalent case behind Equation 5.4.

Summing Equation 5.17 across items for a subject i,

$$X_i = \underbrace{\textstyle\sum_j \nu_j}_{\text{constant}} + \Big(\textstyle\sum_j \lambda_j\Big)\theta_i \;+\; \textstyle\sum_j \varepsilon_{ij}.$$

The baselines $\sum_j \nu_j$ are identical for every model, so they shift all scores equally and add nothing to *between-model* variance. With $\theta \perp \varepsilon$ and independent errors, the total-score variance splits cleanly into a common-factor (true) part and an error part,

$$\sigma_X^2 = \Big(\textstyle\sum_j \lambda_j\Big)^2 \sigma_\theta^2 \;+\; \textstyle\sum_j \psi_j,$$

and reliability is the true share of it—**McDonald's omega**:

$$\omega = \frac{\big(\sum_j \lambda_j\big)^2 \sigma_\theta^2}{\big(\sum_j \lambda_j\big)^2 \sigma_\theta^2 + \sum_j \psi_j}. \tag{5.18}$$

(Fixing $\sigma_\theta^2 = 1$ for identification gives the textbook form.) The signal is the *squared sum* of loadings—items reinforce one another, so loadings add before squaring—while the noise is the plain sum of uniquenesses.

```
#| label: fm-omega
#| autorun: true
#| fig-cap: "Reliability of an 8-item composite as loadings go from uniform to heterogeneous (a few sharply discriminating items among many dull ones), holding the true reliability fixed. McDonald's $\\omega$ tracks the truth throughout; Cronbach's $\\alpha$ matches it only at equal loadings and increasingly understates as they spread, because it assumes a diagnosticity the items do not share."
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.RandomState(0)
N, M = 600, 8
psi     = np.full(M, 0.8)                                   # item uniquenesses
extreme = np.array([1.8, 1.8, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3])   # a few sharp items, many dull
base    = extreme.mean()                                    # uniform loadings, same mean (same true reliability)

```

```
def cronbach_alpha(X):
    k = X.shape[1]
    return k / (k - 1) * (1 - X.var(0, ddof=1).sum() / X.sum(1).var(ddof=1))

def omega_hat(X, iters=30):                                  # iterated principal-axis one-factor fit
    C    = np.cov(X.T)
    dvar = np.diag(C).copy()
    h2   = np.clip(dvar - 1.0 / np.diag(np.linalg.inv(C)), 1e-6, dvar)   # SMC communality start
    Cr   = C.copy()
    for _ in range(iters):                                   # peel uniqueness off the diagonal, refit
        Cr[np.diag_indices_from(Cr)] = h2
        d, V = np.linalg.eigh(Cr)
        lam  = V[:, -1] * np.sqrt(max(d[-1], 1e-9))
        h2   = np.clip(lam ** 2, 1e-6, dvar)
    lam     = np.abs(lam)
    psi_hat = np.clip(dvar - lam ** 2, 1e-9, None)
    return lam.sum() ** 2 / (lam.sum() ** 2 + psi_hat.sum())

contrasts = np.linspace(0, 1, 11)
alpha_v, omega_v, true_v = [], [], []
for c in contrasts:
    lam   = (1 - c) * base + c * extreme                     # uniform -> heterogeneous, sum preserved
    theta = rng.normal(0, 1, N)
    X     = lam[None, :] * theta[:, None] + rng.normal(0, 1, (N, M)) * np.sqrt(psi)[None, :]
    true_v.append(lam.sum() ** 2 / (lam.sum() ** 2 + psi.sum()))
    alpha_v.append(cronbach_alpha(X))
    omega_v.append(omega_hat(X))

plt.figure(figsize=(3.4, 2.2))
plt.plot(contrasts, true_v, 'k--', lw=1, label='true reliability')
plt.plot(contrasts, omega_v, 'o-', color='#2ca02c', ms=3, label="McDonald $\\omega$")
plt.plot(contrasts, alpha_v, 's-', color='#d62728', ms=3, label="Cronbach $\\alpha$")
plt.xlabel('loading heterogeneity'); plt.ylabel('reliability'); plt.ylim(0.5, 0.9)
plt.legend(fontsize=6); plt.grid(True, alpha=0.3); plt.tight_layout(); plt.show()
print(f"uniform loadings : alpha={alpha_v[0]:.3f}, omega={omega_v[0]:.3f}, true={true_v[0]:.3f}")
print(f"hetero. loadings : alpha={alpha_v[-1]:.3f}, omega={omega_v[-1]:.3f}, true={true_v[-1]:.3f}")
```

Heterogeneous loadings are the rule on AI benchmarks—a handful of items separate frontier models while most are saturated—so α systematically understates reliability and ω, which models each item's diagnosticity, is the coefficient to prefer. A bifactor refinement, ω_h, isolates the share carried by a *single general* factor, quantifying how nearly unidimensional a benchmark is—the question α cannot answer (Section 5.10).

5.9 Generalizability for Ordinal Response

Whenever scoring cannot be automated—creative writing, safety, instruction following—a *rater* supplies the score, and the rater facet of Section 5.6 turns from a footnote into the dominant source of variability. Two rater settings recur in modern AI evaluation: an **LLM judge** issuing categorical or pairwise verdicts, and a human or model assigning an **ordinal rating** on a scale. The generalizability machinery of this chapter governs both; each adds its own practical wrinkles—systematic biases for the judge, scale usage and unequal spacing for the rating.

Judges need not return a binary or pairwise verdict. Much of modern AI evaluation is **ordinal**: a judge rates helpfulness on a 1–5 scale, a rubric scores an answer 0–4, MT-Bench grades a response from 1 to 10, a human marks fluency *poor / fair / good / excellent*. The categories are ordered but not equally spaced—the gap from *poor* to *fair* need not equal the gap from *good* to *excellent*. Reliability for such scores needs one extra idea, because the obvious move—average the numbers and decompose their variance—silently assumes the spacing is uniform. Two routes handle ordinal data, trading rigor for convenience.

Treating the scale as interval. Classical generalizability theory was built for numeric ratings, so the simplest route takes the categories at face value: assign them scores $1, \dots, K$, treat the gaps as equal, and run the variance decomposition of Section 5.6 unchanged. For models (p) scored by judges (r),

$$X_{pr} = \mu + \nu_p + \nu_r + \nu_{pr} + \varepsilon, \tag{5.19}$$

with the generalizability coefficient $G = \sigma_p^2/(\sigma_p^2 + \sigma_\delta^2)$ exactly as before. This is what most G-studies do in practice, and it is usually defensible when K is moderate ($K \geq 5$) and the categories are roughly evenly spaced. The cost is the equal-spacing assumption, and a specific artifact it invites: a judge who simply *uses the high end of the scale*—awarding 4s and 5s where another gives 2s and 3s—inflates the judge variance σ_r^2 as pure **scale-usage** bias, even when the two judges rank models identically. Such a main effect hurts *absolute* generalizability (the level of a score) but not *relative* generalizability (the ranking), and the D-study (Section 5.6) reads off how many judges wash it out.

```
#| label: ord-gstudy
#| autorun: true
#| fig-cap: "Ordinal D-study on simulated 1--5 judge ratings with a scale-usage
 ↪  effect. Relative generalizability (ranking models) climbs quickly with more
 ↪  judges; absolute generalizability lags because it must also average out each
 ↪  judge's scale-usage bias $\\sigma_r^2$."
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.RandomState(1)
N, R, K = 120, 6, 5                                  # models, judges, rating categories
 ↪  1..5
```

```
theta    = rng.normal(0, 1.1, N)                  # latent quality of each model
lenience = rng.normal(0, 0.6, R)                  # judge leniency: a scale-usage main effect
cut      = np.array([-2.0, -0.7, 0.7, 2.0])       # K-1 shared category thresholds

def rate(i, r):                                   # ordinal rating in 1..K
    eta = theta[i] + lenience[r] + rng.normal(0, 0.4)   # quality + judge leniency + noise
    return 1 + int(np.sum(eta > cut))
X = np.array([[rate(i, r) for r in range(R)] for i in range(N)], float)

gm  = X.mean()                                    # two-way random-effects ANOVA components
MSp = R * ((X.mean(1) - gm) ** 2).sum() / (N - 1)
MSr = N * ((X.mean(0) - gm) ** 2).sum() / (R - 1)
resid = X - X.mean(1, keepdims=True) - X.mean(0, keepdims=True) + gm
MSe = (resid ** 2).sum() / ((N - 1) * (R - 1))
s2_p = max((MSp - MSe) / R, 0)                     # between-model signal
s2_r = max((MSr - MSe) / N, 0)                     # judge scale usage
s2_e = MSe                                         # interaction + residual

nr = np.arange(1, R + 1)
G_rel = s2_p / (s2_p + s2_e / nr)                  # ranking models
G_abs = s2_p / (s2_p + (s2_r + s2_e) / nr)         # also pay for scale usage

plt.figure(figsize=(3.4, 2))
plt.plot(nr, G_rel, 'o-', color='#2ca02c', label='relative (ranking)')
plt.plot(nr, G_abs, 's--', color='#d62728', label='absolute')
plt.axhline(0.8, color='gray', ls=':', lw=1)
plt.xlabel('number of judges $n_r$'); plt.ylabel('generalizability $G$')
plt.title('Ordinal D-study'); plt.legend(fontsize=6); plt.grid(True, alpha=0.3)
plt.tight_layout(); plt.show()
print(f"variance: model={s2_p:.2f}, judge scale-usage={s2_r:.2f}, residual={s2_e:.2f}")
print(f"G(1 judge): rel={G_rel[0]:.2f}, abs={G_abs[0]:.2f};  G({R} judges): rel={G_rel[-1]:.2f}, abs={G_abs[-1]:.2f}")
```

A latent threshold model. The model-based route keeps the latent decomposition of Section 5.2 and adds ordered cut points. Let the latent quality be $\eta_{ij} = \theta_i - \beta_j + \gamma_{ij}$ (plus rater and occasion facets), with $\mathrm{Var}(\eta) = \sigma_p^2 + \sigma_i^2 + \cdots$ exactly as on the log-odds scale before. A set of ordered thresholds $\tau_1 < \cdots < \tau_{K-1}$ then carves η (plus logistic noise) into categories, giving the **graded-response** or **cumulative-link** model (Samejima 1969; McCullagh 1980):

$$P(Y_{ij} \geq k) = \sigma(\eta_{ij} - \tau_k), \qquad k = 1, \ldots, K-1. \tag{5.20}$$

The thresholds are *where a rater draws the line* between a 3 and a 4, so a judge reluctant to award 5s simply has high upper thresholds—scale usage now lives in τ, a property of the rater, instead

of contaminating the merit θ. Reliability is the same signal share, $\rho = \sigma_p^2 / \operatorname{Var}(\eta)$, computed on the latent scale, and the multinomial spread across categories is the residual—the ordinal analog of the Bernoulli residual of Section 5.5. (The Rasch family has its own ordinal members, the partial-credit and rating-scale models (Masters 1982), for thresholds shared across items.) This route is more work but respects ordinality and isolates scale usage cleanly; Route 1 is its fast approximation.

Agreement for ordinal verdicts. The plain κ of Section 5.9 ignores *how far apart* two ratings are: a 1-vs-2 disagreement counts the same as a 1-vs-5. For ordered categories use **weighted kappa** (Cohen 1968), which penalizes each disagreement by its distance on the scale (linear or quadratic weights). Quadratic-weighted kappa is, conveniently, an estimate of the intraclass correlation—so it is again the rater-facet generalizability coefficient, now respecting order—and Krippendorff's α with an ordinal metric does the same for many raters and missing data. Unweighted κ understates ordinal agreement; the weighted version recovers it.

```
#| label: ord-weighted-kappa
#| autorun: true
#| fig-cap: "Two judges rating 500 items on a 1--5 scale. Unweighted $\\kappa$ counts
 ↪  every disagreement equally and understates agreement; linear- and
 ↪  quadratic-weighted $\\kappa$ forgive near-misses, and quadratic weights track the
 ↪  Pearson correlation---i.e., the intraclass correlation, the rater-facet
 ↪  generalizability coefficient."
N_items, K = 500, 5
rng  = np.random.RandomState(2)
qual = rng.normal(0, 1, N_items)                  # true item quality
cut  = np.array([-1.5, -0.5, 0.5, 1.5])

def judge(noise):                                 # each judge: quality + own noise ->
 ↪  1..K
    return 1 + np.array([int(np.sum(q + rng.normal(0, noise) > cut)) for q in qual])
A, B = judge(0.7), judge(0.7)

def kappa(a, b, K, w='unweighted'):
    O = np.zeros((K, K))
    for x, y in zip(a, b):
        O[x - 1, y - 1] += 1
    O /= O.sum()
    E = np.outer(O.sum(1), O.sum(0))
    i = np.arange(K)
    if   w == 'quadratic': W = (i[:, None] - i[None, :]) ** 2 / (K - 1) ** 2
    elif w == 'linear':    W = np.abs(i[:, None] - i[None, :]) / (K - 1)
    else:                  W = (i[:, None] != i[None, :]).astype(float)
    return 1 - (W * O).sum() / (W * E).sum()

ks   = {w: kappa(A, B, K, w) for w in ['unweighted', 'linear', 'quadratic']}
corr = np.corrcoef(A, B)[0, 1]

plt.figure(figsize=(3.2, 2))
```

```
plt.bar(list(ks) + ['Pearson $r$'], list(ks.values()) + [corr],
        color=['#9467bd', '#1f77b4', '#2ca02c', '#ff7f0e'], alpha=0.85)
plt.ylabel('agreement'); plt.ylim(0, 1); plt.title('Ordinal agreement')
plt.tick_params(labelsize=6); plt.grid(True, alpha=0.3, axis='y'); plt.tight_layout();
 ↪  plt.show()
for w, v in ks.items():
    print(f"{w:>10} kappa = {v:.3f}")
print(f"Pearson r = {corr:.3f}  (quadratic-weighted kappa tracks this)")
```

For open-ended tasks—summarization, safety, instruction following—the dominant modern rater is an **LLM judge** (Zheng et al. 2023), scoring outputs against a rubric or making pairwise preference calls. Everything above applies unchanged: a judge is a rater with its own leniency (γ_r), model-specific quirks (σ^2_{pr}), and run-to-run variability, so aggregating $n_r > 1$ judges buys reliability at the $1/n_r$ rate of the D-study. Two things, though, are special to judges.

Agreement is categorical. Judge outputs are labels or pairwise winners. As flagged with the rater facet in Section 5.2, their agreement is summarized by a chance-corrected agreement coefficient rather than a variance ratio. Three questions recur—do different judges agree (*inter-judge*), does a judge agree with humans (*human–LLM*), is a judge self-consistent across repeats (*intra-judge*)—and each is such a coefficient between the relevant raters.

DEFINITION: COHEN'S KAPPA

For two raters labeling the same items, let p_o be the proportion on which they agree and p_e the agreement expected if each labeled at random from their own marginal frequencies. Cohen's kappa (Cohen 1960) is the share of the *non-chance* agreement that is actually achieved:

$$\kappa = \frac{p_o - p_e}{1 - p_e}.$$

Chance agreement p_e is the noise floor and perfect agreement ($p_o = 1$) the ceiling, so κ runs from 0 (no better than chance) to 1 (perfect)—the agreement counterpart of the generalizability coefficient. Negative values signal systematic disagreement.

Simulate five judges of differing quality on 200 pairwise comparisons, and their pairwise κ shows how much they agree beyond chance.

```
#| label: lj-kappa
#| autorun: true
#| fig-cap: "Pairwise Cohen's $\\kappa$ among five simulated LLM judges of differing
 ↪  quality. Higher-quality judges agree more; the off-diagonal values are the
 ↪  chance-corrected agreement."
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
```

```
N_pairs, N_judges = 200, 5
true_labels = (np.random.normal(0, 1, N_pairs) > 0).astype(int)   # 1 if the first response is better

judge_quality = [0.85, 0.80, 0.75, 0.78, 0.72]   # P(correct)
judge_bias    = [0.0, 0.05, -0.05, 0.10, -0.03]  # position bias

judgments = np.zeros((N_judges, N_pairs), dtype=int)
for j in range(N_judges):
    for k in range(N_pairs):
        judgments[j, k] = true_labels[k] if np.random.rand() < judge_quality[j] else 1 - true_labels[k]
        if np.random.rand() < abs(judge_bias[j]):
            judgments[j, k] = 1 if judge_bias[j] > 0 else 0

def cohen_kappa(a, b):
    p_o = np.mean(a == b)
    pa, pb = a.mean(), b.mean()
    p_e = pa * pb + (1 - pa) * (1 - pb)
    return (p_o - p_e) / (1 - p_e) if p_e < 1 else 0.0

K = np.array([[1.0 if i == j else cohen_kappa(judgments[i], judgments[j])
               for j in range(N_judges)] for i in range(N_judges)])

plt.figure(figsize=(2.8, 2.3))
im = plt.imshow(K, cmap='RdYlGn', vmin=0, vmax=1)
plt.xticks(range(N_judges), [f'J{i+1}' for i in range(N_judges)])
plt.yticks(range(N_judges), [f'J{i+1}' for i in range(N_judges)])
for i in range(N_judges):
    for j in range(N_judges):
        plt.text(j, i, f'{K[i, j]:.2f}', ha='center', va='center', fontsize=6)
plt.title('Inter-judge $\\kappa$'); plt.colorbar(im, shrink=0.8); plt.tight_layout(); plt.show()
print(f"mean inter-judge kappa = {K[np.triu_indices(N_judges, 1)].mean():.3f}")
```

One coefficient for many raters. The matrix above already shows the limit of Cohen's κ: with R judges it has $\binom{R}{2}$ entries and no single summary. Two coefficients collapse it to one number while turning the knobs Cohen's κ holds fixed. **Fleiss's κ** (Fleiss 1971) keeps the chance-corrected form but averages agreement over all rater *pairs* per item, so it handles any number of raters at once. **Krippendorff's α** (Krippendorff 2011) is the general case: writing the observed and expected *disagreement* under a category-distance metric δ,

$$\alpha = 1 - \frac{D_o}{D_e}, \tag{5.21}$$

it recovers the rest by the choice of δ (nominal $\to$ unweighted; interval or ordinal $\to$ weighted), accepts any number of raters, and—because it sums over whatever rating pairs are actually

present—is the only one that survives **missing data**. All four are the same coefficient with three knobs:

Coefficient	Raters	Category metric	Missing data
Cohen's κ	2	nominal	no
Weighted κ	2	ordinal / interval	no
Fleiss's κ	any	nominal	no
Krippendorff's α	any	any (δ)	yes

```
#| label: ord-multirater
#| autorun: true
#| fig-cap: "Four chance-corrected coefficients on the same four-rater ordinal ratings. Fleiss's $\\kappa$ and nominal Krippendorff's $\\alpha$ land together (both credit only exact matches); switching Krippendorff to an interval metric credits near-misses and rises; and after deleting 15% of ratings at random, Krippendorff's $\\alpha$ is essentially unchanged while Fleiss's $\\kappa$ is undefined---it needs complete data."
import numpy as np

rng = np.random.RandomState(3)
U, R, Kc = 150, 4, 5                                # items, raters, categories 0..4
truth = rng.randint(0, Kc, U)
data  = np.clip(np.round(truth[None, :] + rng.normal(0, 0.9, (R, U))), 0, Kc - 1)   # off-by-one ordinal noise

def fleiss_kappa(d):                                # needs complete data
    d = np.asarray(d, int); R, U = d.shape; cats = np.unique(d)
    Nmat = np.stack([(d == c).sum(0) for c in cats], axis=1).astype(float)
    P_i  = ((Nmat ** 2).sum(1) - R) / (R * (R - 1))
    pj   = Nmat.sum(0) / (U * R); Pe = (pj ** 2).sum()
    return (P_i.mean() - Pe) / (1 - Pe)

def kripp_alpha(d, metric='interval'):              # 1 - observed/expected disagreement
    d = np.asarray(d, float); R, U = d.shape
    g = lambda v: (v[:, None] != v[None, :]).astype(float) if metric == 'nominal' else (v[:, None] - v[None, :]) ** 2.0
    num_o = 0.0; n = 0; used = []
    for u in range(U):
        col = d[:, u][~np.isnan(d[:, u])]; m = len(col)
        if m < 2: continue
        vals, cnt = np.unique(col, return_counts=True)
        num_o += (np.outer(cnt, cnt) * g(vals)).sum() / (m - 1); n += m; used.append(col)
    used = np.concatenate(used); vals, cnt = np.unique(used, return_counts=True)
    De = (np.outer(cnt, cnt) * g(vals)).sum() / (n * (n - 1))
    return 1 - (num_o / n) / De

```

```
fk, kn, ki = fleiss_kappa(data), kripp_alpha(data, 'nominal'), kripp_alpha(data, 'interval')
miss = data.astype(float).copy(); miss[rng.rand(R, U) < 0.15] = np.nan      # 15% missing at random
ki_m = kripp_alpha(miss, 'interval')

import matplotlib.pyplot as plt
plt.figure(figsize=(3.6, 2))
plt.bar(["Fleiss\n$\\kappa$", "Kripp.\nnominal", "Kripp.\ninterval", "Kripp. int.\n15% missing"],
        [fk, kn, ki, ki_m], color=['#9467bd', '#1f77b4', '#2ca02c', '#2ca02c'], alpha=0.85)
plt.ylabel('agreement'); plt.ylim(0, 1); plt.tick_params(labelsize=6)
plt.grid(True, alpha=0.3, axis='y'); plt.tight_layout(); plt.show()
print(f"Fleiss kappa = {fk:.3f};  Krippendorff nominal = {kn:.3f}  (many raters, nominal metric)")
print(f"Krippendorff interval = {ki:.3f}  (credits near-misses);  with 15% missing = {ki_m:.3f}")
```

Fleiss's κ and nominal Krippendorff's α land together; the interval metric lifts the coefficient by crediting near-misses (the multi-rater echo of weighted κ); and only Krippendorff's α still returns a number once ratings go missing—which is why it is the default for messy, multi-annotator evaluation data.

Aggregating judges. Because a judge is a rater, the D-study remedy applies directly: pool several independent judges by majority vote and accuracy climbs at the $1/n_r$ rate, just as adding raters raised G above.

```
#| label: lj-majority
#| autorun: true
#| fig-cap: "Accuracy of a majority vote over an increasing number of judges. Pooling independent judges sharpens the verdict---the D-study, applied to the rater facet."
counts = range(1, N_judges + 1)
acc = []
for n_j in counts:
    samples = [np.mean(((judgments[np.random.choice(N_judges, n_j, replace=False)].mean(0) > 0.5)
                        .astype(int)) == true_labels) for _ in range(100)]
    acc.append(np.mean(samples))

plt.figure(figsize=(3.2, 2))
plt.plot(list(counts), acc, 'o-', color='steelblue', lw=1.5, ms=5)
plt.xlabel('number of judges'); plt.ylabel('accuracy'); plt.xticks(list(counts))
plt.title('Majority-vote accuracy'); plt.grid(True, alpha=0.3); plt.tight_layout(); plt.show()
print(f"single judge = {acc[0]:.3f},  5-judge majority = {acc[-1]:.3f}")
```

Bias is not noise. Everything so far—CTT, G-theory, agreement coefficients—measures *random* error, which averages out with more data. Judges also have *systematic* errors that do not: position bias (favoring the response shown first (Wang et al. 2023)), length preference, self-preference. These shift the verdict, not its variance, so they are invisible to reliability.

> **! Systematic vs. random error**
>
> Reliability analysis detects only *random* error—zero-mean, averaging away with replication. A systematic error like position bias is consistent, so a biased judge can be perfectly *reliable* yet *invalid*, and high inter-judge agreement does not rescue it: two judges can share the same bias. On a balanced benchmark such bias is even invisible to aggregate accuracy, because favoring "first" helps exactly as often as it hurts. Detecting it requires the validity tools of Chapter 1, or a design trick: swap the presentation order. A position-biased judge then contradicts itself, and the contradiction rate rises with the bias.

```
#| label: lj-position-bias
#| autorun: true
#| fig-cap: "Order-inconsistency rate (the judge's two verdicts disagree when the responses are swapped) against position-bias strength. Bias is invisible to a single-order run but shows up as self-contradiction across orders---which is also why averaging both orders cancels it."
np.random.seed(0)
N, q = 4000, 0.80
t = (np.random.rand(N) > 0.5).astype(int)        # 1 if the canonical-first response is better

def says_first(first_better, b):                  # does the judge call the displayed-first better?
    v = first_better if np.random.rand() < q else 1 - first_better
    return 1 if np.random.rand() < b else v       # position bias overrides toward "first"

bias_levels = np.linspace(0, 0.3, 16)
inconsistency = [sum(says_first(t[k], b) == says_first(1 - t[k], b) for k in range(N)) / N
                 for b in bias_levels]

plt.figure(figsize=(3.2, 2))
plt.plot(bias_levels, inconsistency, '-', color='#d62728', lw=1.8)
plt.xlabel('position-bias strength'); plt.ylabel('order-inconsistency rate')
plt.title('Bias exposed by swapping order'); plt.grid(True, alpha=0.3)
plt.tight_layout(); plt.show()
print(f"inconsistency: no bias = {inconsistency[0]:.3f}, strong bias = {inconsistency[-1]:.3f}")
```

Designing a judge protocol. The remedies combine the D-study with bias control: aggregate several independent judges; randomize the presentation order and average, cancelling position bias into noise; score at temperature 0 to remove sampling stochasticity; write concrete rubrics,

which shrink σ_r^2 and σ_{ir}^2; and calibrate against a human gold set, since agreement alone cannot catch a bias the judges share.

5.10 Generalizability for CTT

The estimators so far—plug-in (Section 5.3), Bayesian (Section 5.4), and method of moments (Section 5.5)—read the variance components off the full response matrix. Classical test theory (CTT), their early-twentieth-century antecedent, asks for less: it recovers reliability from *total scores alone*, with no fitted item parameters. It keeps the same decomposition but bundles every non-person term into a single *error*, writing the observed score $X = T + E$ with true score $T_i = \mathbb{E}[X_i]$, $\sigma_T^2 = V_{\text{person}}$, and $\sigma_E^2 = V_{\text{item}} + V_{\text{interaction}} + V_{\text{residual}}$. Reliability is the true-score share $\rho_{XX'} = \sigma_T^2/\sigma_X^2$—for a single graded response exactly the person share of Equation 5.13, the same object the response model and generalizability theory estimate, now read off the scores rather than fitted.

Because true scores are unobservable, $\rho_{XX'}$ must be inferred from how scores covary, and the classical coefficients are just different cuts of that one quantity. They also settle a puzzle from Section 5.5: a *single* graded response had reliability near 0.09, yet hundred-item benchmarks rank models stably. The reconciliation is aggregation, made exact by the **Spearman–Brown** length law—a test of K parallel length-M forms with per-form reliability ρ has reliability

$$\rho_K = \frac{K\rho}{1 + (K-1)\rho}, \tag{5.22}$$

so reliability climbs predictably as items accumulate (a 50-item benchmark at $\rho = 0.80$ reaches 0.89 at 100 items and 0.94 at 200): averaging many noisy items multiplies signal over noise.

Three coefficients estimate the *same* $\rho_{XX'}$ from different data structures, each trading what it assumes for what it requires:

- **Test–retest** correlates scores from two occasions—re-running at temperature > 0, or re-scoring the same outputs—so the Pearson correlation estimates $\rho_{XX'}$, but folds every occasion-to-occasion source into one number.
- **Split-half** correlates two half-tests and rescales the half-length correlation by Spearman–Brown, $\rho_{XX'} = 2r_{12}/(1 + r_{12})$, needing no re-run.
- **Cronbach's** α averages over *all* splits at once,

$$\alpha = \frac{M}{M-1}\left(1 - \frac{\sum_{j=1}^{M}\sigma_j^2}{\sigma_X^2}\right), \tag{5.23}$$

with σ_j^2 the variance of item j and σ_X^2 the total-score variance—the internal-consistency coefficient, and (from Section 5.8) the tau-equivalent special case of McDonald's ω.

What Cronbach's Alpha Does and Does Not Reveal

Alpha measures how much items covary relative to total variance. High alpha means the items are measuring something in common. But alpha does *not* indicate: (a) whether that common thing is unidimensional, (b) whether that thing is the intended construct, or (c) whether the test is free of systematic bias.

A benchmark with $\alpha = 0.95$ might be reliably measuring training data contamination rather than the intended capability. High reliability is necessary but not sufficient for good measurement.

Simulate a 100-model, 100-item Rasch benchmark: split-half correlates two half-test scores, then Spearman–Brown rescales to the full length.

```
#| label: rc-split-half
#| autorun: true
#| fig-cap: "Split-half reliability on a simulated 100-item Rasch benchmark: the two
 ↪  half-test scores correlate strongly, and the Spearman--Brown correction rescales
 ↪  the half-length correlation to the full test."
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return np.where(x >= 0, 1 / (1 + np.exp(-x)), np.exp(x) / (1 + np.exp(x)))

np.random.seed(42)
N, M = 100, 100                                    # models, items
theta = np.random.normal(0, 1, N)
beta  = np.random.normal(0, 1.5, M)
Y = (np.random.rand(N, M) < sigmoid(theta[:, None] - beta[None, :])).astype(float)

half1, half2 = Y[:, :M // 2].mean(1), Y[:, M // 2:].mean(1)
r12    = np.corrcoef(half1, half2)[0, 1]
rho_sb = 2 * r12 / (1 + r12)                        # Spearman-Brown correction

plt.figure(figsize=(3.2, 2))
plt.scatter(half1, half2, alpha=0.5, s=15, color='steelblue')
plt.plot([0, 1], [0, 1], 'k--', alpha=0.3)
plt.xlabel('half 1 score'); plt.ylabel('half 2 score')
plt.title(f'Split-half ($\\rho$ = {rho_sb:.3f})'); plt.grid(True, alpha=0.3)
plt.tight_layout(); plt.show()
```

Cronbach's α averages over all possible splits; as items accumulate it climbs along the Spearman–Brown curve predicted from a short test.

```
#| label: rc-alpha
#| autorun: true
```

```
#| fig-cap: "Cronbach's $\\alpha$ against the number of items, with the Spearman--Brown prediction extrapolated from the five-item value. Reliability rises with test length exactly as the prophecy formula anticipates."
item_counts = np.arange(5, M + 1, 5)
alphas = []
for m in item_counts:
    Ys = Y[:, :m]
    alphas.append((m / (m - 1)) * (1 - Ys.var(0, ddof=1).sum() / Ys.sum(1).var(ddof=1)))

base    = alphas[0]
K       = item_counts / item_counts[0]
sb_pred = K * base / (1 + (K - 1) * base)           # Spearman-Brown from the 5-item value

plt.figure(figsize=(3.2, 2))
plt.plot(item_counts, alphas, 'o-', color='steelblue', ms=3, label='empirical $\\alpha$')
plt.plot(item_counts, sb_pred, '--', color='coral', label='Spearman-Brown')
plt.xlabel('number of items'); plt.ylabel("Cronbach's $\\alpha$")
plt.title('Reliability vs test length'); plt.ylim(0, 1)
plt.legend(fontsize=7); plt.grid(True, alpha=0.3); plt.tight_layout(); plt.show()
print(f"alpha: {M} items = {alphas[-1]:.3f};  Spearman-Brown from 5 items = {sb_pred[-1]:.3f}")
```

From reliability to an error bar. A coefficient earns its keep once turned into uncertainty on each score. The **standard error of measurement** does exactly that, shrinking the spread of observed scores by the fraction that is noise:

$$\mathrm{SEM} = \sigma_X \sqrt{1 - \rho_{XX'}}.$$

A benchmark with score spread $\sigma_X = 5$ points and $\rho_{XX'} = 0.90$ has SEM ≈ 1.58 points, so a 95% interval around any model spans $\pm 2\,\mathrm{SEM} \approx \pm 3.2$ points: two models closer than that may swap ranks on a replication—the usual situation on crowded leaderboards where dozens of models cluster within a few points. As reliability rises with test length, the SEM shrinks in step.

```
#| label: rc-sem
#| autorun: true
#| fig-cap: "Standard error of measurement against the number of items. As reliability climbs the SEM falls, narrowing the confidence band around each model's score."
sigma_X = np.array([Y[:, :m].mean(1).std() for m in item_counts])
sems    = sigma_X * np.sqrt(1 - np.array(alphas))

plt.figure(figsize=(3.2, 2))
plt.plot(item_counts, sems, 'o-', color='steelblue', ms=3)
```

```
plt.xlabel('number of items'); plt.ylabel('SEM (score units)')
plt.title('Standard error of measurement'); plt.grid(True, alpha=0.3)
plt.tight_layout(); plt.show()
print(f"full test ({M} items): alpha = {alphas[-1]:.3f}, SEM = {sems[-1]:.4f}")
```

5.11 Item-Level Diagnostics

The diagnostics in this section read structure out of the same variance decomposition that defines reliability. Recall the person-by-item interaction γ_{ij} of Section 5.2, whose canonical example was a question "idiosyncratically easy for one model family": once that interaction is indexed by a *meaningful group*, the same term becomes a bias to detect rather than a variance to average down—which is exactly what differential item functioning does. The tools below—DIF, dimensionality assessment, item fit, and the multitrait-multimethod matrix—operationalize the validity threats catalogued conceptually in Section 1.3: each is a structured reading of how items depart from a single, shared construct.

5.11.1 Differential Item Functioning

Differential Item Functioning (DIF) occurs when an item is systematically easier or harder for one subgroup of examinees than for another, *after controlling for the overall ability level*. In classical psychometrics, DIF analysis compares demographic groups; in AI evaluation, the "groups" might be model families, architectural types, or training paradigms.

Formally, an item i exhibits DIF if:

$$P(X_{ij} = 1 \mid \theta_j, g = 1) \neq P(X_{ij} = 1 \mid \theta_j, g = 0) \tag{5.24}$$

where θ_j is model j's ability and g is group membership. The conditioning on θ_j is critical: we are asking whether the item behaves differently for equally able models from different groups.

DIF comes in two forms:

- **Uniform DIF**: The item is consistently easier (or harder) for one group across all ability levels. The ICCs for the two groups are shifted but do not cross.
- **Non-uniform DIF**: The item favors one group at some ability levels and the other group at different ability levels. The ICCs cross.

The **Mantel-Haenszel (MH) procedure** (Holland and Wainer 1993) is the most widely used DIF detection method. It stratifies examinees by ability (using total score as a proxy), computes odds ratios at each stratum, and combines them into a summary statistic:

$$\alpha_{MH} = \frac{\sum_k A_k D_k / N_k}{\sum_k B_k C_k / N_k} \tag{5.25}$$

where A_k, B_k, C_k, D_k are the entries of the 2×2 table (correct/incorrect $\times$ group) at ability stratum k, and N_k is the stratum size. An odds ratio of 1 indicates no DIF; values significantly different from 1 indicate DIF.

AI example. Consider a mathematics benchmark comparing open-source and proprietary models. Some items might rely on LaTeX-formatted mathematical expressions that proprietary models handle better due to training data composition, independent of mathematical reasoning ability. These items would exhibit DIF: the formatting advantage inflates scores for one group without reflecting the target construct.

5.11.2 DIF Analysis

The Mantel-Haenszel procedure described in Section 5.11.1 provides a nonparametric DIF test. A complementary approach uses **logistic regression** (Zumbo 1999):

$$\log \frac{P(X_{ij} = 1)}{P(X_{ij} = 0)} = \beta_0 + \beta_1 \theta_j + \beta_2 g_j + \beta_3 (\theta_j \times g_j) \tag{5.26}$$

where θ_j is the ability proxy (total score), g_j is group membership, and β_3 captures the interaction. Testing $\beta_2 = 0$ detects uniform DIF; testing $\beta_3 = 0$ detects non-uniform DIF.

The following simulation demonstrates DIF analysis on a benchmark with planted DIF items.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: dif-analysis
#| autorun: true
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.special import expit
from scipy.stats import chi2

np.random.seed(42)

# --- Simulation setup ---
n_models = 200  # 100 per group
n_items = 30
n_dif_items = 5  # items with planted DIF

# Model abilities: two groups with same ability distribution
theta_g0 = np.random.normal(0, 1, 100)  # Group 0 (e.g., open-source)
theta_g1 = np.random.normal(0, 1, 100)  # Group 1 (e.g., proprietary)
theta = np.concatenate([theta_g0, theta_g1])
group = np.array([0]*100 + [1]*100)

# Item difficulties
beta = np.linspace(-2, 2, n_items)

# Generate responses: Rasch model with DIF on last 5 items
# DIF items are 0.8 logits easier for Group 1
dif_shift = np.zeros(n_items)
dif_shift[-n_dif_items:] = -0.8  # negative = easier for Group 1

X = np.zeros((n_models, n_items))
for j in range(n_models):
    for i in range(n_items):
        shift = dif_shift[i] if group[j] == 1 else 0
        p = expit(theta[j] - beta[i] + shift)
        X[j, i] = np.random.binomial(1, p)

# --- Mantel-Haenszel DIF detection ---
total_scores = X.sum(axis=1)
# Create 5 ability strata based on total score quintiles
strata = np.digitize(total_scores, np.percentile(total_scores, [20, 40, 60, 80]))

mh_odds_ratios = []
mh_chi2 = []

for i in range(n_items):
    num = 0; den = 0
    a_tot = 0; b_tot = 0; c_tot = 0; d_tot = 0
    for k in range(5):
        mask = (strata == k)
        g0_mask = mask & (group == 0)
        g1_mask = mask & (group == 1)

        a = X[g0_mask, i].sum()  # Group 0, correct
```

```
        b = g0_mask.sum() - a     # Group 0, incorrect
        c = X[g1_mask, i].sum()  # Group 1, correct
        d = g1_mask.sum() - c     # Group 1, incorrect

        Nk = g0_mask.sum() + g1_mask.sum()
        if Nk > 0:
            num += (a * d) / Nk
            den += (b * c) / Nk
        a_tot += a; b_tot += b; c_tot += c; d_tot += d

    alpha_mh = num / den if den > 0 else np.inf
    mh_odds_ratios.append(alpha_mh)

    # MH chi-square (simplified)
    log_or = np.log(alpha_mh) if alpha_mh > 0 and alpha_mh < np.inf else 0
    mh_chi2.append(log_or)

mh_log_or = np.array([np.log(x) if 0 < x < np.inf else 0 for x in mh_odds_ratios])

# --- Visualization ---
fig, axes = plt.subplots(1, 2, figsize=(6, 2))

# Panel 1: MH log-odds ratios
colors = ['#E8637A' if i >= n_items - n_dif_items else '#5B8DEE' for i in
 ↪  range(n_items)]
axes[0].bar(range(n_items), mh_log_or, color=colors, alpha=0.8, edgecolor='white')
axes[0].axhline(y=0, color='black', linewidth=0.8)
axes[0].axhline(y=np.log(1.5), color='gray', linewidth=0.8, linestyle='--',
 ↪  label='|log OR| = 0.41 threshold')
axes[0].axhline(y=-np.log(1.5), color='gray', linewidth=0.8, linestyle='--')
axes[0].set_xlabel('Item Index')
axes[0].set_ylabel('MH Log-Odds Ratio')
axes[0].set_title('DIF Detection: Mantel-Haenszel')
from matplotlib.patches import Patch
axes[0].legend(handles=[
    Patch(facecolor='#5B8DEE', label='Non-DIF items'),
    Patch(facecolor='#E8637A', label='DIF items (planted)'),
    plt.Line2D([0], [0], color='gray', linestyle='--', label='Detection threshold')
])

# Panel 2: ICC curves for a DIF item vs non-DIF item
theta_range = np.linspace(-3, 3, 100)

# Non-DIF item (item 10)
p_g0_nondif = expit(theta_range - beta[10])
p_g1_nondif = expit(theta_range - beta[10])

# DIF item (item 28)
p_g0_dif = expit(theta_range - beta[28])
```

```
p_g1_dif = expit(theta_range - beta[28] + dif_shift[28])

axes[1].plot(theta_range, p_g0_nondif, '#5B8DEE', linewidth=2, label='Non-DIF: Group 0')
axes[1].plot(theta_range, p_g1_nondif, '#5B8DEE', linewidth=2, linestyle='--', label='Non-DIF: Group 1')
axes[1].plot(theta_range, p_g0_dif, '#E8637A', linewidth=2, label='DIF: Group 0')
axes[1].plot(theta_range, p_g1_dif, '#E8637A', linewidth=2, linestyle='--', label='DIF: Group 1')
axes[1].set_xlabel(r'Ability ($\theta$)')
axes[1].set_ylabel('P(Correct)')
axes[1].set_title('ICC Comparison: DIF vs Non-DIF Item')
axes[1].legend()
axes[1].set_ylim(-0.05, 1.05)

plt.tight_layout()
plt.show()

# Summary
n_flagged = sum(1 for x in mh_log_or if abs(x) > np.log(1.5))
print(f"\nDIF Analysis Summary:")
print(f"  Items flagged (|log OR| > 0.41): {n_flagged} / {n_items}")
print(f"  True DIF items: {n_dif_items}")
dif_detected = sum(1 for i in range(n_items - n_dif_items, n_items) if abs(mh_log_or[i]) > np.log(1.5))
print(f"  Correctly detected: {dif_detected} / {n_dif_items}")
```

The left panel shows the Mantel-Haenszel log-odds ratio for each item. Items near zero show no DIF; items with large absolute values are flagged. The planted DIF items (red) cluster at the extremes, while non-DIF items (blue) remain near zero. The right panel compares Item Characteristic Curves: for a non-DIF item, the two groups share the same ICC; for a DIF item, Group 1's curve is shifted leftward (the item is easier for them at every ability level), indicating uniform DIF.

In practice, DIF analysis for AI benchmarks would compare model families (e.g., autoregressive vs. encoder-decoder), training paradigms (e.g., RLHF vs. base models), or architectural features (e.g., mixture-of-experts vs. dense transformers).

5.11.3 Dimensionality Assessment

A benchmark that claims to measure a single construct (e.g., "reasoning") should exhibit a *unidimensional* factor structure: one latent factor explaining the bulk of the inter-item covariance. Verifying that claim—and, more generally, recovering *how many* constructs an item set measures—is the discovery question developed in Section 7.1, where **parallel analysis** (Horn 1965) compares the observed eigenvalues of the item correlation matrix against those from random data of the same shape and retains a factor only when its eigenvalue exceeds the random-data threshold. The use here is confirmatory: if the unidimensionality that a single composite score presupposes

fails the test, the score is conflating multiple constructs, and the remedy is to report subscores per factor or revise the benchmark toward one construct.

5.11.4 Item Fit and Contamination Detection

Item-fit statistics from Rasch modeling (Wright and Masters 1982) measure how well individual items conform to the measurement model. Items that misfit—those whose response patterns deviate from model predictions—may indicate contamination, construct-irrelevant features, or other validity problems.

The two standard fit statistics are:

- **Infit** (information-weighted mean square): Sensitive to unexpected responses near an item's difficulty level. Infit emphasizes on-target responses and is less affected by extreme scores.
- **Outfit** (unweighted mean square): Sensitive to unexpected responses far from an item's difficulty level. Large outfit values indicate that the item behaves erratically for very high- or very low-ability models.

For a Rasch model with predicted probability $P_{ij} = \text{expit}(\theta_j - \beta_i)$, the standardized residual is:

$$z_{ij} = \frac{X_{ij} - P_{ij}}{\sqrt{P_{ij}(1 - P_{ij})}}$$

The outfit mean square for item i is:

$$\text{Outfit}_i = \frac{1}{N} \sum_{j=1}^{N} z_{ij}^2$$

and infit weights by the variance:

$$\text{Infit}_i = \frac{\sum_{j=1}^{N} (X_{ij} - P_{ij})^2}{\sum_{j=1}^{N} P_{ij}(1 - P_{ij})}$$

Expected value under the model is 1.0. Values above 1.3 suggest underfit (more noise than the model predicts, possibly multidimensionality or noise). Values below 0.7 suggest overfit (too predictable, possibly item redundancy or dependency). Very high values (above 2.0) indicate serious misfit.

Contamination leaves a distinctive item-fit signature: contaminated items become much easier for models that have memorized them (producing unexpectedly correct responses from low-ability models) while remaining at their true difficulty for uncontaminated models. This

inflates outfit statistics because the misfitting responses come from models far from the item's difficulty.

```
#| label: item-fit-contamination
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit

np.random.seed(321)

# --- Simulation: Rasch benchmark with contaminated items ---
n_models = 150
n_items = 25
n_contaminated = 5  # Items that some models have memorized

theta = np.random.normal(0, 1, n_models)
beta = np.linspace(-2, 2, n_items)

# Mark 30 models as "contaminated" (saw some test items during training)
contaminated_models = np.random.choice(n_models, 30, replace=False)
contaminated_items = list(range(n_items - n_contaminated, n_items))

# Generate responses
X = np.zeros((n_models, n_items))
for j in range(n_models):
    for i in range(n_items):
        p = expit(theta[j] - beta[i])
        # Contaminated models get contaminated items right with high probability
        if j in contaminated_models and i in contaminated_items:
            p = 0.95  # Near-certain correct due to memorization
        X[j, i] = np.random.binomial(1, p)

# --- Estimate item parameters (simple MLE for Rasch) ---
# Use proportion correct as rough difficulty proxy
p_obs = X.mean(axis=0)
beta_hat = -np.log(p_obs / (1 - p_obs + 1e-10))

# Use total score as ability proxy
total_scores = X.sum(axis=1)
theta_hat = (total_scores - total_scores.mean()) / (total_scores.std() + 1e-10)

# --- Compute infit and outfit ---
infit = np.zeros(n_items)
outfit = np.zeros(n_items)

for i in range(n_items):
    P = expit(theta_hat - beta_hat[i])
    variance = P * (1 - P)
```

```
    residual = X[:, i] - P
    z_sq = residual**2 / (variance + 1e-10)

    outfit[i] = z_sq.mean()
    infit[i] = (residual**2).sum() / (variance.sum() + 1e-10)

# --- Visualization ---
fig, axes = plt.subplots(1, 2, figsize=(6, 2))

# Panel 1: Infit vs Outfit scatter
colors = ['#E8637A' if i in contaminated_items else '#5B8DEE' for i in range(n_items)]
axes[0].scatter(infit, outfit, c=colors, s=80, edgecolors='white', linewidth=0.5,
 ↪  zorder=3)
for i in range(n_items):
    axes[0].annotate(f'{i+1}', (infit[i] + 0.02, outfit[i] + 0.02), fontsize=7,
 ↪  color='gray')

# Acceptable range
axes[0].axhline(y=1.3, color='gray', linewidth=0.8, linestyle='--')
axes[0].axhline(y=0.7, color='gray', linewidth=0.8, linestyle='--')
axes[0].axvline(x=1.3, color='gray', linewidth=0.8, linestyle='--')
axes[0].axvline(x=0.7, color='gray', linewidth=0.8, linestyle='--')
axes[0].axhline(y=1, color='gray', linewidth=0.5, linestyle=':')
axes[0].axvline(x=1, color='gray', linewidth=0.5, linestyle=':')

from matplotlib.patches import Patch
axes[0].legend(handles=[
    Patch(facecolor='#5B8DEE', label='Clean items'),
    Patch(facecolor='#E8637A', label='Contaminated items')
], loc='upper left')
axes[0].set_xlabel('Infit MNSQ')
axes[0].set_ylabel('Outfit MNSQ')
axes[0].set_title('Item Fit Statistics')

# Panel 2: Observed vs expected ICC for a contaminated item
item_idx = contaminated_items[2]  # Pick one contaminated item
theta_sorted = np.argsort(theta_hat)
theta_bins = np.array_split(theta_sorted, 10)

obs_prop = []
exp_prop = []
theta_mid = []
for bin_idx in theta_bins:
    obs_prop.append(X[bin_idx, item_idx].mean())
    exp_prop.append(expit(theta_hat[bin_idx] - beta_hat[item_idx]).mean())
    theta_mid.append(theta_hat[bin_idx].mean())

theta_smooth = np.linspace(-3, 3, 100)
icc_expected = expit(theta_smooth - beta[item_idx])
```

```
94
95  axes[1].plot(theta_smooth, icc_expected, '#5B8DEE', linewidth=2, label='Expected
     ↪  (Rasch)')
96  axes[1].scatter(theta_mid, obs_prop, color='#E8637A', s=80, zorder=3,
97                  edgecolors='white', linewidth=0.5, label='Observed proportions')
98  axes[1].set_xlabel(r'Ability ($\theta$)')
99  axes[1].set_ylabel('P(Correct)')
100 axes[1].set_title(f'Item {item_idx + 1} (Contaminated): Observed vs Expected')
101 axes[1].legend()
102 axes[1].set_ylim(-0.05, 1.05)
103
104 plt.tight_layout()
105 plt.show()
106
107 # Summary
108 print(f"\nItem Fit Summary:")
109 print(f"  Items with outfit > 1.3: {sum(1 for x in outfit if x > 1.3)}")
110 print(f"  Items with infit > 1.3: {sum(1 for x in infit if x > 1.3)}")
111 contaminated_flagged = sum(1 for i in contaminated_items if outfit[i] > 1.3 or
     ↪  infit[i] > 1.3)
112 print(f"  Contaminated items flagged: {contaminated_flagged} / {n_contaminated}")
113 print(f"\n  Contaminated item fit values:")
114 for i in contaminated_items:
115     print(f"    Item {i+1}: infit={infit[i]:.2f}, outfit={outfit[i]:.2f}")
```

The left panel shows the infit-outfit scatter: clean items (blue) cluster near the expected value of 1.0, while contaminated items (red) show inflated fit statistics, especially outfit. This is because contaminated models produce unexpectedly correct responses on these items, creating large residuals. The right panel shows the ICC for a single contaminated item: the observed proportions (red dots) consistently exceed the Rasch-expected curve (blue line), particularly at lower ability levels where the memorization effect is most visible.

Item-fit analysis is a powerful diagnostic because it does not require knowing *which* models are contaminated or *which* items are compromised. The Rasch model's prediction residuals reveal anomalies regardless of their source.

5.11.5 The Multitrait-Multimethod Matrix

The Multitrait-Multimethod (MTMM) matrix, introduced by Campbell and Fiske (1959), provides a systematic framework for assessing both convergent and discriminant validity simultaneously. The idea is to measure multiple *traits* (constructs) using multiple *methods* (measurement formats) and examine the resulting correlation structure.

In AI evaluation:

- **Traits** are the constructs of interest: reasoning, knowledge retrieval, creativity, instruction following, etc.

- **Methods** are the benchmark formats: multiple-choice questions, open-ended generation, pairwise comparison, code completion, etc.

The MTMM matrix organizes correlations into four blocks:

1. **Monotrait-heteromethod** (same trait, different methods): These should be high—this is convergent validity. If "reasoning" measured by MCQ and by open-ended generation produces highly correlated scores, the construct is robust across methods.
2. **Heterotrait-monomethod** (different traits, same method): These should be lower than monotrait-heteromethod correlations—this is discriminant validity. If "reasoning-MCQ" correlates as highly with "knowledge-MCQ" as with "reasoning-open-ended," then method variance dominates trait variance.
3. **Heterotrait-heteromethod** (different traits, different methods): These should be lowest—neither shared trait nor shared method.
4. **Reliability diagonals** (same trait, same method, repeated): These should be highest—they set the ceiling for validity.

Campbell and Fiske's criteria for construct validity require:

1. Convergent validity coefficients (monotrait-heteromethod) should be significantly different from zero and large enough to warrant further investigation.
2. Convergent validity coefficients should be higher than the values in the same row and column of the heterotrait-heteromethod block.
3. Convergent validity coefficients should be higher than the heterotrait-monomethod correlations.
4. The pattern of trait intercorrelations should be consistent across method blocks.

```
#| label: mtmm-matrix
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(456)

# --- Simulate MTMM data: 3 traits x 2 methods ---
n_models = 200
n_traits = 3
n_methods = 2
trait_names = ['Reasoning', 'Knowledge', 'Creativity']
method_names = ['MCQ', 'Open-ended']

# Latent trait scores (moderately correlated)
trait_corr = np.array([
    [1.0, 0.3, 0.2],
    [0.3, 1.0, 0.25],
    [0.2, 0.25, 1.0]
])
```

```
L_trait = np.linalg.cholesky(trait_corr)
traits = np.random.normal(0, 1, (n_models, n_traits)) @ L_trait.T

# Method effects (shared variance due to format)
method_effects = np.random.normal(0, 0.5, (n_models, n_methods))

# Observed scores: trait + method effect + noise
scores = np.zeros((n_models, n_traits * n_methods))
labels = []
for t in range(n_traits):
    for m in range(n_methods):
        col = t * n_methods + m
        trait_loading = 0.7 + np.random.uniform(-0.05, 0.05)
        method_loading = 0.3 + np.random.uniform(-0.05, 0.05)
        noise = np.random.normal(0, 0.3, n_models)
        scores[:, col] = trait_loading * traits[:, t] + method_loading *
 ↪  method_effects[:, m] + noise
        labels.append(f'{trait_names[t]}\n({method_names[m]})')

# --- Compute MTMM correlation matrix ---
corr = np.corrcoef(scores.T)

# --- Visualization ---
fig, axes = plt.subplots(1, 2, figsize=(6, 2))

# Panel 1: Full MTMM matrix as heatmap
im = axes[0].imshow(corr, cmap='RdBu_r', vmin=-1, vmax=1, aspect='equal')
axes[0].set_xticks(range(len(labels)))
axes[0].set_yticks(range(len(labels)))
axes[0].set_xticklabels(labels, rotation=45, ha='right')
axes[0].set_yticklabels(labels)

# Annotate values
for i in range(len(labels)):
    for j in range(len(labels)):
        color = 'white' if abs(corr[i, j]) > 0.6 else 'black'
        axes[0].text(j, i, f'{corr[i, j]:.2f}', ha='center', va='center',
                     fontsize=6, color=color)

# Draw block boundaries
for k in range(1, n_traits):
    pos = k * n_methods - 0.5
    axes[0].axhline(y=pos, color='black', linewidth=1.5)
    axes[0].axvline(x=pos, color='black', linewidth=1.5)

axes[0].set_title('MTMM Correlation Matrix')
plt.colorbar(im, ax=axes[0], fraction=0.046, pad=0.04)

# Panel 2: Summary of validity coefficients
```

```
# Extract convergent (monotrait-heteromethod) correlations
convergent = []
for t in range(n_traits):
    i, j = t * n_methods, t * n_methods + 1
    convergent.append(corr[i, j])

# Extract discriminant (heterotrait-monomethod) correlations
discriminant_mono = []
for m in range(n_methods):
    for t1 in range(n_traits):
        for t2 in range(t1 + 1, n_traits):
            i = t1 * n_methods + m
            j = t2 * n_methods + m
            discriminant_mono.append(corr[i, j])

# Extract heterotrait-heteromethod correlations
discriminant_hetero = []
for t1 in range(n_traits):
    for t2 in range(n_traits):
        if t1 != t2:
            for m1 in range(n_methods):
                for m2 in range(n_methods):
                    if m1 != m2:
                        i = t1 * n_methods + m1
                        j = t2 * n_methods + m2
                        discriminant_hetero.append(corr[i, j])

categories = ['Convergent\n(monotrait-\nheteromethod)',
              'Discriminant\n(heterotrait-\nmonomethod)',
              'Discriminant\n(heterotrait-\nheteromethod)']
means = [np.mean(convergent), np.mean(discriminant_mono),
 ↪  np.mean(discriminant_hetero)]
stds = [np.std(convergent) if len(convergent) > 1 else 0,
        np.std(discriminant_mono), np.std(discriminant_hetero)]

bar_colors = ['#45BF7C', '#F0A35C', '#E8637A']
bars = axes[1].bar(categories, means, yerr=stds, color=bar_colors, alpha=0.8,
                   edgecolor='white', capsize=5)
axes[1].set_ylabel('Mean Correlation')
axes[1].set_title('Validity Coefficient Summary')
axes[1].set_ylim(0, 1)

# Add value labels
for bar, mean in zip(bars, means):
    axes[1].text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.03,
                 f'{mean:.2f}', ha='center', fontweight='bold')

plt.tight_layout()
plt.show()
```

```
117
118 print(f"\nMTMM Validity Check:")
119 print(f"  Convergent validity (mean): {np.mean(convergent):.3f}")
120 print(f"  Discriminant - monomethod (mean): {np.mean(discriminant_mono):.3f}")
121 print(f"  Discriminant - heteromethod (mean): {np.mean(discriminant_hetero):.3f}")
122
123 # Campbell-Fiske criteria
124 cf1 = all(c > 0 for c in convergent)
125 cf2 = np.mean(convergent) > np.mean(discriminant_hetero)
126 cf3 = np.mean(convergent) > np.mean(discriminant_mono)
127 print(f"\n  Campbell-Fiske Criteria:")
128 print(f"    1. Convergent > 0: {'PASS' if cf1 else 'FAIL'}")
129 print(f"    2. Convergent > heterotrait-heteromethod: {'PASS' if cf2 else 'FAIL'}")
130 print(f"    3. Convergent > heterotrait-monomethod: {'PASS' if cf3 else 'FAIL'}")
```

The heatmap (left) shows the full MTMM correlation matrix with trait blocks separated by black lines. Strong convergent validity appears as high correlations along the off-diagonal within each trait block (same trait measured by different methods). The bar chart (right) confirms the expected ordering: convergent validity coefficients (green) are highest, discriminant-monomethod correlations (orange) are moderate (reflecting shared method variance), and discriminant-heteromethod correlations (red) are lowest. This ordering satisfies the Campbell-Fiske criteria for construct validity.

For AI evaluation, constructing an MTMM study requires measuring the same constructs with genuinely different methods. If all benchmarks use the same multiple-choice format, discriminant validity cannot be assessed because method variance is confounded with trait variance.

5.12 Building Trustworthy AI Evaluations

Having developed tools to *detect* validity problems (Section 5.11), we turn to *building* benchmarks with validity designed in from the start. Those tools double as construction aids: the multitrait-multimethod analysis (Section 5.11.5) supplies evidence for the nomological network by testing whether the expected convergent and discriminant relationships hold, and the item-level diagnostics of Section 5.11.2 and Section 5.11.4 confirm that individual items behave as the construct theory predicts.

5.12.1 Principled Item Construction and Revision

Constructing a valid benchmark requires more than assembling a large set of items. The process should be systematic:

1. **Define the construct**: Write an explicit construct definition specifying what the benchmark is intended to measure and what it is *not* intended to measure. This definition serves as the foundation for content validity.

2. **Specify the domain**: Create a detailed content specification (sometimes called a "test blueprint" or "table of specifications") that enumerates the sub-domains, difficulty levels, and item types that should be represented.
3. **Generate candidate items**: Items can be written by domain experts, sourced from existing materials, or generated synthetically using LLMs. For synthetic generation, the construct definition and difficulty targets serve as prompts.
4. **Pilot and analyze**: Administer items to a representative sample of models. Compute item statistics: difficulty (p-value or IRT β), discrimination (a parameter or point-biserial correlation), and fit (infit/outfit from Section 5.11.4).
5. **Screen for validity threats**: Run DIF analysis (Section 5.11.2) to check for group bias. Run dimensionality analysis (Section 7.1) to verify unidimensionality (if claimed). Check for contamination using fit statistics and public/private splits.
6. **Revise and iterate**: Remove or revise items with poor statistics. Replace them with new items and re-pilot. This cycle continues until the item pool meets quality standards.

The following simulation illustrates the pilot-and-analyze phase.

```
#| label: item-construction
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit

np.random.seed(789)

# --- Simulate item pool with varying quality ---
n_models = 200
n_candidate_items = 40

theta = np.random.normal(0, 1, n_models)

# Item parameters: some good, some problematic
beta = np.random.uniform(-2.5, 2.5, n_candidate_items)  # Difficulty
alpha = np.zeros(n_candidate_items)  # Discrimination

# Good items: discrimination 0.8-1.5
good_items = list(range(0, 30))
alpha[good_items] = np.random.uniform(0.8, 1.5, len(good_items))

# Poor items: very low discrimination (nearly random)
poor_disc_items = list(range(30, 35))
alpha[poor_disc_items] = np.random.uniform(0.05, 0.25, len(poor_disc_items))

# Contaminated items: appear easy due to memorization
contaminated = list(range(35, 40))
```

```
alpha[contaminated] = np.random.uniform(0.8, 1.2, len(contaminated))
# These items have artificially high correct rates

# Generate responses (2PL model)
X = np.zeros((n_models, n_candidate_items))
for j in range(n_models):
    for i in range(n_candidate_items):
        p = expit(alpha[i] * (theta[j] - beta[i]))
        # Contaminated items: 40% of models get them right regardless
        if i in contaminated and np.random.random() < 0.4:
            p = 0.95
        X[j, i] = np.random.binomial(1, p)

# --- Compute item statistics ---
# Difficulty: proportion correct
p_values = X.mean(axis=0)

# Point-biserial correlation (discrimination proxy)
total_scores = X.sum(axis=1)
point_biserial = np.array([np.corrcoef(X[:, i], total_scores)[0, 1] for i in
 ↪  range(n_candidate_items)])

# Infit (simplified Rasch-based)
# Rough ability estimates from total score
theta_hat = (total_scores - total_scores.mean()) / (total_scores.std() + 1e-10)
beta_hat = -np.log(p_values / (1 - p_values + 1e-10))

infit = np.zeros(n_candidate_items)
for i in range(n_candidate_items):
    P = expit(theta_hat - beta_hat[i])
    variance = P * (1 - P)
    residual = X[:, i] - P
    infit[i] = (residual**2).sum() / (variance.sum() + 1e-10)

# --- Flag items ---
flags = []
for i in range(n_candidate_items):
    item_flags = []
    if point_biserial[i] < 0.2:
        item_flags.append('Low discrimination')
    if infit[i] > 1.3:
        item_flags.append('Misfit')
    if infit[i] < 0.7:
        item_flags.append('Overfit')
    if p_values[i] > 0.95 or p_values[i] < 0.05:
        item_flags.append('Extreme difficulty')
    flags.append(item_flags)

# --- Visualization ---
```

```
fig, axes = plt.subplots(1, 3, figsize=(6, 2))

# Assign colors based on item type
def get_color(i):
    if i in poor_disc_items: return '#E8637A'
    if i in contaminated: return '#F0A35C'
    return '#5B8DEE'

item_colors = [get_color(i) for i in range(n_candidate_items)]

# Panel 1: Difficulty vs Discrimination
axes[0].scatter(p_values, point_biserial, c=item_colors, s=60,
                edgecolors='white', linewidth=0.5, zorder=3)
axes[0].axhline(y=0.2, color='gray', linewidth=0.8, linestyle='--', label='Min
 ↪  discrimination')
axes[0].axvline(x=0.05, color='gray', linewidth=0.8, linestyle=':')
axes[0].axvline(x=0.95, color='gray', linewidth=0.8, linestyle=':')
axes[0].set_xlabel('Difficulty (Proportion Correct)')
axes[0].set_ylabel('Point-Biserial Correlation')
axes[0].set_title('Item Difficulty vs Discrimination')
from matplotlib.patches import Patch
axes[0].legend(handles=[
    Patch(facecolor='#5B8DEE', label='Good items'),
    Patch(facecolor='#E8637A', label='Low discrimination'),
    Patch(facecolor='#F0A35C', label='Contaminated')
], loc='lower left')

# Panel 2: Infit distribution
axes[1].bar(range(n_candidate_items), infit, color=item_colors, alpha=0.8,
 ↪  edgecolor='white')
axes[1].axhline(y=1.0, color='black', linewidth=0.8)
axes[1].axhline(y=1.3, color='gray', linewidth=0.8, linestyle='--')
axes[1].axhline(y=0.7, color='gray', linewidth=0.8, linestyle='--')
axes[1].fill_between(range(n_candidate_items), 0.7, 1.3, alpha=0.05, color='green')
axes[1].set_xlabel('Item Index')
axes[1].set_ylabel('Infit MNSQ')
axes[1].set_title('Item Fit Statistics')

# Panel 3: Summary of item screening
n_good = sum(1 for f in flags if len(f) == 0)
n_flagged = sum(1 for f in flags if len(f) > 0)
categories = ['Accepted', 'Flagged']
counts = [n_good, n_flagged]
bar_colors_summary = ['#45BF7C', '#E8637A']
bars = axes[2].bar(categories, counts, color=bar_colors_summary, alpha=0.8,
 ↪  edgecolor='white')
for bar, count in zip(bars, counts):
    axes[2].text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.5,
                 str(count), ha='center', fontweight='bold')
```

```
axes[2].set_ylabel('Number of Items')
axes[2].set_title('Item Screening Results')

plt.tight_layout()
plt.show()

print(f"\nItem Screening Summary:")
print(f"  Total candidate items: {n_candidate_items}")
print(f"  Accepted (no flags): {n_good}")
print(f"  Flagged for review: {n_flagged}")
print(f"\n  Flag breakdown:")
flag_counts = {}
for f_list in flags:
    for f in f_list:
        flag_counts[f] = flag_counts.get(f, 0) + 1
for flag, count in sorted(flag_counts.items()):
    print(f"    {flag}: {count} items")
```

The three panels show the item screening workflow. The difficulty-discrimination plot (left) reveals that low-discrimination items (red) cluster below the 0.2 threshold, providing little measurement information, while contaminated items (orange) show inflated proportion-correct values. The infit plot (center) highlights items that deviate from model predictions. The summary (right) shows the yield: how many candidate items survive screening. Items that are flagged should be revised or replaced before the benchmark is finalized.

5.12.2 Item-Design Principles

We close with six design principles that synthesize the concepts developed in this chapter.

1. **Start with the construct, not the data**. Define what is being measured before collecting items. Write an explicit construct definition and content specification. This prevents construct underrepresentation and makes content validity assessable.

2. **Use multiple methods**. Measure each construct with at least two different formats (e.g., MCQ and open-ended). This enables MTMM analysis and separates trait variance from method variance. A construct that is robust across methods has stronger validity evidence.

3. **Check dimensionality**. Before reporting a single composite score, verify that the benchmark is unidimensional (or report subscores for each dimension). Parallel analysis is cheap and can prevent misleading composites.

4. **Screen for DIF**. Whenever the model population includes distinct subgroups (architectural families, training paradigms, model sizes), run DIF analysis to detect items that function differently across groups.

5. **Monitor for contamination**. Maintain a held-out private item pool. Compare performance on public and private items. Use item-fit statistics to flag items where some models perform unexpectedly well.

6. **Document validity evidence**. Follow the claim-centered approach: for each claim the benchmark is intended to support, document the available validity evidence and its limitations. This transparency allows users to judge whether the benchmark supports *their* intended interpretation.

Worked example: evaluating summarization ability. Suppose the goal is to build a benchmark for evaluating how well language models summarize documents.

- *Construct definition*: The ability to produce concise, accurate, and coherent summaries of source documents, preserving key information while omitting irrelevant details.
- *Content specification*: Documents from 5 domains (news, scientific, legal, medical, conversational), 3 lengths (short, medium, long), 2 genres (informative, narrative).
- *Multiple methods*: Evaluate summaries with (a) ROUGE against reference summaries, (b) LLM-as-judge on a rubric, (c) human expert ratings.
- *Dimensionality check*: Factor analysis reveals two dimensions: factual accuracy and coherence. Report subscores for each.
- *DIF analysis*: Compare models fine-tuned for summarization vs. general-purpose models. Flag items where domain-specific vocabulary creates an irrelevant advantage.
- *Contamination check*: Use held-out documents not available online. Compare private vs. public document performance gaps.

5.13 Design principles

The theory developed in this chapter yields concrete design principles:

1. **Identify the dominant source of variability** (Section 5.6). Run a G-study before committing to a design. If rater variability dominates, invest in more raters or clearer rubrics; if item-sampling variability dominates, use more items or D-optimal selection.
2. **Match items to the ability range of interest** (Section 5.3). Conditional reliability varies across ability levels; ranking frontier models needs items hard enough to discriminate among them.
3. **Use multiple judges when scoring is subjective** (Section 5.9). The D-study quantifies the gain: three independent judges with majority vote are substantially more reliable than one, and often cheaper than more items.
4. **Report confidence intervals, not just point estimates** (Section 5.10). Two models whose scores overlap within ± 2 SEM may not be meaningfully different; leaderboards should mark which ranking differences are reliable.
5. **Control what can be controlled** (Section 5.1). Score at temperature 0, randomize and average over prompt formats, standardize rubrics—these *eliminate* sources of variability rather than averaging over them.

6. **Separate reliability from validity** (Chapter 1). High reliability is necessary but not sufficient: a benchmark can be reliable while systematically measuring the wrong thing (contamination, shortcut features, length bias).
7. **Audit items, don't just score them** (Section 3.3). Reliability diagnostics flag broken benchmark items—negative item–total correlations signal flipped keys, ambiguity, or grading bugs.

5.14 Bibliographic Notes

Classical test theory. The foundational reference is Lord and Novick (1968), which formalized CTT axiomatically. Cronbach (1951) introduced coefficient alpha, the most widely used reliability statistic. The Spearman–Brown formula dates to Spearman (1910) and Brown (1910) independently. For the connection between CTT and IRT-based reliability, see Chapter 7 of Hambleton and Swaminathan (1985).

Variance decomposition and component estimation. The decomposition of a function's variance into additive main-effect and interaction terms—the *functional ANOVA* this chapter builds on—was introduced for U-statistics by Hoeffding (1948) and, in the global-sensitivity literature, by Sobol (2001); Efron and Stein (1981) give the closely related variance bound. Estimating the resulting variance components from data is the subject of Searle et al. (1992): the method-of-moments (expected mean squares) route traces to Henderson (1953), and restricted maximum likelihood (REML)—the modern default for unbalanced designs—to Patterson and Thompson (1971).

Generalizability theory. G-theory was developed by Cronbach et al. (1972), building on earlier work by Cronbach on the multiple sources of measurement error. Brennan (2001) provides the comprehensive modern treatment. Shavelson and Webb (1991) offers an accessible primer. The connection between G-theory and mixed-effects models makes modern software (e.g., `lme4` in R) directly applicable to G-studies.

Inter-rater reliability. Cohen (1960) introduced Cohen's kappa. Fleiss (1971) extended it to multiple raters. Krippendorff (2011) provides Krippendorff's alpha, which handles missing data, ordinal scales, and any number of raters. For a comprehensive treatment with practical guidance, see Gwet (2014).

Model selection. For choosing a facet structure by predictive accuracy, Watanabe (2010) introduced the widely applicable information criterion (WAIC) and Vehtari et al. (2017) the practical PSIS-LOO cross-validation estimate; both approximate held-out likelihood while integrating out random effects (Section 5.7).

Factor-analytic reliability. McDonald's ω derives from the common-factor model (McDonald 1999); Revelle and Zinbarg (2009) compare α, ω, and the greatest lower bound, and develop ω_h for general-factor saturation. The lower-bound family in which Cronbach's α is one member (λ_3) traces to Guttman (1945) (Section 5.8).

Ordinal and graded responses. The graded-response model is due to Samejima (1969); McCullagh (1980) developed the cumulative-link (proportional-odds) regression that underlies it, and Masters (1982) the partial-credit Rasch variant. Weighted kappa for ordered categories is Cohen (1968); with quadratic weights it estimates the intraclass correlation, linking ordinal agreement back to the generalizability coefficient (Section 5.9).

LLM-as-a-judge. Zheng et al. (2023) introduced the LLM-as-a-judge paradigm and the MT-Bench evaluation framework. Position bias was documented by Wang et al. (2023). Shankar et al. (2024) addresses the question of validating LLM judges against human preferences. For prompt sensitivity in evaluation, see Mizrahi et al. (2024). Biderman et al. (2024) discusses reproducibility challenges in language model evaluation more broadly.

Item-level diagnostics. The Multitrait-Multimethod matrix was introduced by Campbell and Fiske (1959) and remains one of the most systematic approaches to convergent and discriminant validity. Differential Item Functioning methods are covered comprehensively by Holland and Wainer (1993) and Zumbo (1999). Item-fit statistics for Rasch models are developed by Wright and Masters (1982), and parallel analysis for dimensionality assessment by Horn (1965).

5.15 Exercises

The first six exercises are conceptual; the rest involve derivation or implementation, marked ⋆ (straightforward) to ⋆ ⋆ ⋆ (challenging).

1. **Determinism and reliability**. At temperature 0, a language model gives the same output every time. Does this mean the evaluation has perfect test–retest reliability? What sources of unreliability remain even with deterministic models?

2. **Reliability vs. number of items**. Spearman–Brown predicts that longer tests are more reliable, and AI benchmarks often have thousands of items—far more than typical educational tests. Does this make them automatically reliable? What assumptions might be violated?

3. **LLM-as-a-judge tradeoffs**. Using an LLM as a judge is cheaper than human annotation but introduces model-specific biases. Under what conditions is a less accurate but more reliable judge preferable? How does generalizability theory formalize the tradeoff?

4. **Reliability across model generations**. When a new generation of models is released, a benchmark's reliability can change as item difficulties shift relative to model abilities. How should benchmark developers monitor and maintain reliability over time?

5. **The reliability–validity tension**. A benchmark of 1000 copies of the same easy question would have extremely high internal consistency but near-zero validity. How does this illustrate the distinction, and what design principles prevent it?

6. **The item-removal paradox**. Spearman–Brown says shortening a test *lowers* reliability, yet dropping an item that correlates poorly with the rest can *raise* Cronbach's α. Reconcile these: when does removing an item help reliability on net, and what does that reveal

about the equal-quality (parallel-items) assumption behind Spearman–Brown? (The item-quality diagnostics that flag such items are developed in Section 3.3.)

7. **Deriving reliability** ($\star$). Starting from the CTT model $X = T + E$ with the standard assumptions ($\mathbb{E}[E] = 0$, $\mathrm{Cov}(T, E) = 0$), derive $\rho_{XX'} = \sigma_T^2/\sigma_X^2$, and show reliability equals the squared correlation between observed and true scores, $\rho_{XX'} = r_{XT}^2$.

8. **Spearman–Brown** ($\star\star$). Derive the prophecy formula: a test of length KM built from K parallel forms of a length-M test with reliability ρ has reliability $\rho_K = K\rho/(1+(K-1)\rho)$. What happens as $K \to \infty$?

9. **Generalizability coefficient** ($\star\star$). In a $p \times i \times r$ design, derive the generalizability coefficient G, and show that increasing n_r reduces the contribution of rater variance but not item variance, and vice versa.

10. **Cronbach's α as a mean split-half** ($\star\star$). Show that α is the mean of all possible split-half reliabilities (after Spearman–Brown correction) for an M-item test. Under what conditions is α a lower bound for reliability?

11. **G-study implementation** ($\star\star$). Implement a G-study for a model$\times$item$\times$rater design: given a 3-way array $X[p, i, r]$, estimate all variance components by expected mean squares, and verify on simulated data that the estimates recover the truth.

12. **LLM-judge simulation** ($\star\star$). Simulate an LLM-as-a-judge evaluation with 5 judges of varying quality (agreement with a gold standard from 0.70 to 0.90). Compute (a) pairwise Cohen's κ and (b) majority-vote accuracy for 1, 3, and 5 judges. How does reliability change if all judges share a systematic bias, such as preferring longer responses?

13. **D-study optimizer** ($\star\star\star$). Given variance components and costs per item (c_i) and per rater–item pair (c_r), find the cheapest design (n_i, n_r) achieving $G \geq G^\star$. Solve it as a constrained optimization and visualize the cost surface.

14. **Chatbot Arena reliability** ($\star\star$). The Chatbot Arena uses a single human preference judgment per comparison. Using generalizability theory, analyze the reliability of the resulting Elo ratings: how many comparisons per model pair would reach $G \geq 0.90$, and what are the cost implications?

15. **Ordinal reliability: interval vs. latent** ($\star\star$). Simulate 1–5 judge ratings from the graded-response model of Equation 5.20 with a judge scale-usage effect. Compute the generalizability coefficient (a) by treating the scale as interval (Section 5.9) and (b) from the fitted latent threshold model. When do the two diverge, and how does the gap depend on the number of categories K and the size of the scale-usage effect?

16. **Weighted kappa and the ICC** ($\star\star$). Show that quadratic-weighted kappa equals the intraclass correlation for two raters on an interval scale. Then simulate ordinal ratings and verify numerically that unweighted kappa understates agreement while quadratic-weighted kappa tracks the Pearson correlation between raters. Why does this make quadratic-weighted kappa the natural rater-facet generalizability coefficient for ordinal data?

17. **McDonald's ω from first principles** ($\star\star$). Starting from the congeneric one-factor model $X_{ij} = \nu_j + \lambda_j\theta_i + \varepsilon_{ij}$ (Equation 5.17), derive the total-score reliability $\omega = (\sum_j \lambda_j)^2\sigma_\theta^2/[(\sum_j \lambda_j)^2\sigma_\theta^2 + \sum_j \psi_j]$. Simulate items with unequal loadings and verify numerically that $\alpha < \omega \approx$ the true reliability; then set all loadings equal and confirm ω collapses to the Spearman–Brown step-up of the single-item reliability (Equation 5.4).

18. **Does the interaction earn its place?** ($\star\star\star$). Generate replicated binary responses from a model with a person-by-item interaction ($\sigma_{pi}^2 > 0$) and from one without. For each, fit a main-effects model and a saturated (per-cell) model and compare them by held-out log-likelihood on (a) masked cells and (b) masked trials. Show that the iid interaction is selectable only from held-out *trials*, and only once replication is sufficient (Section 5.7). Why does a held-out-*cell* comparison fail to detect it?

19. **Empirical Bayes vs. full Bayes** ($\star\star$). With one binary trial per cell and abilities $\theta_i \sim \mathcal{N}(0, \sigma_p^2)$, estimate σ_p two ways: by marginal maximum likelihood (the evidence peak) and by reporting the full posterior mean and 95% credible interval under a flat hyperprior (Section 5.4). Explain why the two point estimates differ and why the credible interval is wide when there is only one trial per cell.

Item-level diagnostics. These accompany the diagnostics of Section 5.11.

20. Prove that if an item exhibits uniform DIF between two groups (i.e., the item is uniformly easier for one group at all ability levels), the standard Rasch model $P(X_{ij} = 1) = \text{expit}(\theta_j - \beta_i)$ cannot hold simultaneously for both groups with the same item parameter β_i. What does this imply about the meaning of "item difficulty" when DIF is present?

21. Consider a 2-trait $\times$ 2-method MTMM design. Show that if all convergent validity coefficients equal 1 (perfect convergent validity) and all heterotrait correlations equal 0 (perfect discriminant validity), then the method factors contribute zero variance. What does this imply about the interpretability of scores in practice, where method variance is nonzero?

22. Define infit mean square as $\text{Infit}_i = \sum_j (X_{ij} - P_{ij})^2 / \sum_j P_{ij}(1 - P_{ij})$. Suppose a fraction γ of models have memorized item i and respond correctly with probability 1 regardless of ability. Derive the expected infit as a function of γ and the item difficulty β_i. For what values of γ does the infit exceed the conventional 1.3 threshold?

23. Download a publicly available AI benchmark dataset (e.g., MMLU, HellaSwag, or ARC). Divide the models into two groups (e.g., models above and below 7B parameters). Conduct a Mantel-Haenszel DIF analysis and report which items, if any, show significant DIF. Interpret the results: does the DIF reflect genuine ability differences or construct-irrelevant features?

24. Simulate a benchmark with 3 underlying factors (e.g., verbal, quantitative, and spatial reasoning) using a factor model from Chapter 3. Generate responses for 300 models on 30 items (10 per factor). Implement parallel analysis to determine the correct dimensionality.

How does the accuracy of parallel analysis depend on the factor correlations and sample size?

25. Using the MTMM simulation framework from Section 5.11.5, vary the relative strength of trait loadings vs. method loadings. At what ratio do the Campbell–Fiske criteria begin to fail? Plot the convergent and discriminant validity coefficients as a function of the trait-to-method variance ratio.

26. Design an MTMM study for evaluating "mathematical reasoning" in large language models. Specify the traits, methods, and the expected pattern of convergent and discriminant correlations.

6 Efficiency

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Formulate** the benchmark design problem as an optimization over item selection and scoring rules.
2. **Apply** Fisher information to select maximally informative items, and implement a Computerized Adaptive Testing procedure.
3. **Construct** D-optimal item pools that maximize the precision of ability estimates.
4. **Design** efficient paired-comparison tournaments using information-theoretic principles.
5. **Explain** stopping rules and practical constraints (cost, time, contamination) for adaptive AI evaluation.

Suggested Lecture Plan

This chapter can be covered in **2 lectures** (75-90 minutes each):

Lecture 1: Fisher Information and Adaptive Testing

- The benchmark design problem (15 min)
- Fisher information for item selection (20 min)
- Computerized Adaptive Testing (30 min)
- Hands-on: CAT simulation (10 min)

Lecture 2: Optimal Design and Paired Comparisons

- D-optimal design and item pool construction (30 min)
- Design for paired comparisons (20 min)
- Stopping rules and practical considerations (15 min)
- Hands-on: D-optimal design simulation (10 min)

Notation

This chapter introduces $I_j(\theta)$ (Fisher information for item j) and $\mathcal{I}(\theta)$ (Fisher information matrix). See the notation appendix[1] for the complete notation reference.

[1] notation.qmd

6.1 The Design Problem in AI Evaluation

Chapter 3 introduced the measurement models—Rasch, 2PL, factor models, Bradley-Terry—that formalize how latent abilities generate observed responses. Chapter 4 showed how to estimate the parameters of these models from data. But both chapters took the response matrix Y as given. In practice, someone must *design* the evaluation: choosing which items to include and how to score responses. These design decisions profoundly affect the quality and efficiency of the resulting measurements.

The benchmark designer faces two fundamental efficiency questions:

1. **Item selection**: Which questions or tasks should the benchmark include? Given a pool of candidate items, how do we select a subset that maximizes the precision of our measurements?
2. **Scoring rules**: How do we aggregate responses into scores? Sum scores, weighted scores, latent factor scores? The choice interacts with the measurement model (recall from Chapter 3 that sum scores are sufficient for the Rasch model but not for 2PL).

This chapter develops the statistical foundations for efficient evaluation design, drawing on classical experimental design and information theory. We apply these frameworks to the practical problem of designing AI evaluation benchmarks that achieve high measurement precision with minimal cost.

6.2 Optimal Experimental Design

We begin with the simplest setting: the evaluator wants to measure model abilities as precisely as possible, and the models respond honestly. This is the classical problem of *optimal experimental design* applied to measurement.

6.2.1 Fisher Information for Item Selection

Recall from Chapter 3 that the Fisher information for item j at ability θ in the Rasch model is:

$$I_j(\theta) = P_j(\theta) \cdot (1 - P_j(\theta)) \tag{6.1}$$

where $P_j(\theta) = \sigma(\theta - \beta_j)$ is the probability of a correct response. This quantity measures how much observing a response to item j tells us about θ.

Fisher information is maximized when $P_j(\theta) = 0.5$, which occurs when $\theta = \beta_j$—the item difficulty matches the model's ability. Intuitively, a question that a model gets right 99% of the time or wrong 99% of the time reveals almost nothing about the model's ability. The most informative questions are those where the outcome is uncertain.

For a test consisting of items $\{j_1, \ldots, j_K\}$, the total information is additive under local independence:

$$I_{\text{total}}(\theta) = \sum_{k=1}^{K} I_{j_k}(\theta)$$

This additivity is the foundation of optimal item selection: we want to choose items that collectively maximize the total information across the range of abilities we care about.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: fisher-information-design
#| autorun: true
#| fig-cap: "Fisher information curves illustrate the key design principle: items are
 ↪  most informative when their difficulty matches the test-taker's ability."

import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    """Numerically stable sigmoid function."""
    return np.where(x >= 0,
                    1 / (1 + np.exp(-x)),
                    np.exp(x) / (1 + np.exp(x)))

theta_range = np.linspace(-4, 4, 200)

fig, axes = plt.subplots(1, 1, figsize=(2, 2))

```

```
# Information curves for different item difficulties
difficulties = [-2, -1, 0, 1, 2]
colors = plt.cm.viridis(np.linspace(0, 1, len(difficulties)))

for beta_j, color in zip(difficulties, colors):
    P = sigmoid(theta_range - beta_j)
    info = P * (1 - P)
    axes[0].plot(theta_range, info, color=color, linewidth=2,
                 label=f'Item difficulty = {beta_j}')

axes[0].set_xlabel('Ability ()')
axes[0].set_ylabel('Fisher Information')
axes[0].set_title('Item Information Curves')
axes[0].legend(fontsize=7)
axes[0].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

6.2.2 Computerized Adaptive Testing

Computerized Adaptive Testing (CAT) is the sequential application of optimal item selection. Rather than administering a fixed test to all models, CAT adapts the test in real time: after each response, it selects the next item that would be most informative given what we have learned so far.

The CAT procedure iterates four steps:

1. **Select** the most informative item given the current ability estimate
2. **Administer** the item and observe the response
3. **Update** the ability estimate using the new data
4. **Check** a stopping criterion; if not met, return to step 1

> **! WHY FISHER INFORMATION FOR ITEM SELECTION?**
>
> Fisher information measures how much a response to item j tells us about θ:
>
> – **High information**: The item difficulty is well-matched to the ability level
> – **Low information**: The item is too easy or too hard
>
> Asking a frontier model to answer $1 + 1$ or a small model to prove the Riemann hypothesis provides almost no information. The most informative items are those where the model has roughly a 50% chance of success.

```
#| label: fisher-information-design
#| autorun: true
```

```
#| fig-cap: "Fisher information curves illustrate the key design principle: items are most informative when their difficulty matches the test-taker's ability."

# Cumulative information: adaptive vs random
np.random.seed(42)
N, M = 100, 50
theta_true = np.random.normal(0, 1, N)
beta_true = np.random.normal(0, 1.5, M)

theta_test = 0.5
n_items = 20
fig, axes = plt.subplots(1, 1, figsize=(2, 2))

# Adaptive: choose items closest to current estimate
beta_available = beta_true.copy()
adaptive_info = [0]
theta_estimate = 0

for t in range(n_items):
    distances = np.abs(beta_available - theta_estimate)
    best_idx = np.argmin(distances)
    beta_selected = beta_available[best_idx]
    P = sigmoid(theta_test - beta_selected)
    info = P * (1 - P)
    adaptive_info.append(adaptive_info[-1] + info)
    beta_available = np.delete(beta_available, best_idx)
    theta_estimate = theta_test

# Random selection
random_order = np.random.permutation(len(beta_true))[:n_items]
random_info = [0]
for j in random_order:
    P = sigmoid(theta_test - beta_true[j])
    info = P * (1 - P)
    random_info.append(random_info[-1] + info)

axes[1].plot(range(n_items + 1), adaptive_info, 'g-', linewidth=2, label='Adaptive')
axes[1].plot(range(n_items + 1), random_info, 'b-', linewidth=2, label='Random')
axes[1].set_xlabel('Number of Items')
axes[1].set_ylabel('Cumulative Fisher Information')
axes[1].set_title(f'Information Accumulation (  = {theta_test})')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

6.2.3 CAT Implementation

```
#| label: cat-simulation
#| autorun: true
#| fig-cap: "CAT achieves the same measurement precision as random testing with substantially fewer items."

# Generate synthetic data for CAT simulation
np.random.seed(42)
N, M = 100, 50
theta_true = np.random.normal(0, 1, N)
beta_true = np.random.normal(0, 1.5, M)

def cat_simulation(theta_true_i, beta, n_items_max=30, reliability_threshold=0.95):
    """
    Simulate CAT for a single test-taker.

    Parameters
    ----------
    theta_true_i : float
        True ability of the test-taker
    beta : ndarray
        Item difficulties (pre-calibrated)
    n_items_max : int
        Maximum number of items to administer
    reliability_threshold : float
        Stop when reliability exceeds this threshold

    Returns
    -------
    dict with theta_hat, n_items, reliability_history, etc.
    """
    M = len(beta)
    administered = []
    responses = []

    # Prior: theta ~ N(0, 1)
    theta_hat = 0.0
    prior_var = 1.0

    theta_history = [theta_hat]
    reliability_history = [0.0]
    se_history = [1.0]
    available_items = list(range(M))

    for t in range(min(n_items_max, M)):
        # Select item with maximum Fisher information at current estimate
        best_item = None
        best_info = -np.inf
```

```
47
48          for j in available_items:
49              P_j = sigmoid(theta_hat - beta[j])
50              info_j = P_j * (1 - P_j)
51              if info_j > best_info:
52                  best_info = info_j
53                  best_item = j
54
55          # Administer item (simulate response from true ability)
56          P_true = sigmoid(theta_true_i - beta[best_item])
57          response = int(np.random.random() < P_true)
58
59          administered.append(best_item)
60          responses.append(response)
61          available_items.remove(best_item)
62
63          # Update ability estimate via MAP (Newton-Raphson)
64          for _ in range(10):
65              P_vec = sigmoid(theta_hat - np.array([beta[j] for j in administered]))
66              grad = np.sum(np.array(responses) - P_vec) - theta_hat / prior_var
67              hess = -np.sum(P_vec * (1 - P_vec)) - 1 / prior_var
68              if abs(hess) > 1e-10:
69                  theta_hat = theta_hat - grad / hess
70
71          # Posterior variance and reliability
72          total_info = np.sum([sigmoid(theta_hat - beta[j]) * (1 - sigmoid(theta_hat -
    ↪ beta[j]))
73                               for j in administered])
74          posterior_var = 1 / (1/prior_var + total_info)
75          reliability = 1 - posterior_var / prior_var
76
77          theta_history.append(theta_hat)
78          reliability_history.append(reliability)
79          se_history.append(np.sqrt(posterior_var))
80
81          if reliability >= reliability_threshold:
82              break
83
84      return {
85          'theta_hat': theta_hat, 'n_items': len(administered),
86          'reliability_history': reliability_history,
87          'theta_history': theta_history, 'se_history': se_history,
88          'final_reliability': reliability_history[-1]
89      }
90
91  def random_selection_simulation(theta_true_i, beta, n_items_max=30,
    ↪ reliability_threshold=0.95):
92      """Simulate random item selection for comparison."""
93      M = len(beta)
```

```
    item_order = list(np.random.permutation(M)[:n_items_max])

    theta_hat = 0.0
    prior_var = 1.0
    administered = []
    responses = []
    reliability_history = [0.0]
    theta_history = [theta_hat]

    for j in item_order:
        P_true = sigmoid(theta_true_i - beta[j])
        response = int(np.random.random() < P_true)
        administered.append(j)
        responses.append(response)

        for _ in range(10):
            P_vec = sigmoid(theta_hat - np.array([beta[k] for k in administered]))
            grad = np.sum(np.array(responses) - P_vec) - theta_hat / prior_var
            hess = -np.sum(P_vec * (1 - P_vec)) - 1 / prior_var
            if abs(hess) > 1e-10:
                theta_hat = theta_hat - grad / hess

        total_info = np.sum([sigmoid(theta_hat - beta[k]) * (1 - sigmoid(theta_hat -
↪   beta[k]))
                             for k in administered])
        posterior_var = 1 / (1/prior_var + total_info)
        reliability = 1 - posterior_var / prior_var
        reliability_history.append(reliability)
        theta_history.append(theta_hat)

        if reliability >= reliability_threshold:
            break

    return {'n_items': len(administered), 'reliability_history': reliability_history,
            'theta_history': theta_history, 'theta_hat': theta_hat,
            'final_reliability': reliability_history[-1]}

# Run simulations
np.random.seed(42)
n_test_takers = 100
theta_test_sample = np.random.normal(0, 1, n_test_takers)

cat_results = []
random_results = []
for theta_i in theta_test_sample:
    cat_results.append(cat_simulation(theta_i, beta_true))
    random_results.append(random_selection_simulation(theta_i, beta_true))

cat_items = [r['n_items'] for r in cat_results]
```

```
random_items = [r['n_items'] for r in random_results]

# Plot comparison
fig, axes = plt.subplots(1, 3, figsize=(6, 2))

# Bar chart
methods = ['Random', 'CAT']
means = [np.mean(random_items), np.mean(cat_items)]
stds = [np.std(random_items), np.std(cat_items)]
bars = axes[0].bar(methods, means, yerr=stds, capsize=5, alpha=0.7,
                   color=['#1f77b4', '#2ca02c'])
axes[0].set_ylabel('Items to reach 95% reliability')
axes[0].set_title('Efficiency: CAT vs Random')
axes[0].grid(True, alpha=0.3, axis='y')
for bar, mean, std in zip(bars, means, stds):
    axes[0].text(bar.get_x() + bar.get_width()/2, bar.get_height() + std + 0.5,
                 f'{mean:.1f}', ha='center', va='bottom', fontsize=11)

# Reliability trajectory
example_idx = 50
axes[1].plot(random_results[example_idx]['reliability_history'], 'b-', linewidth=2,
 ↪  label='Random')
axes[1].plot(cat_results[example_idx]['reliability_history'], 'g-', linewidth=2,
 ↪  label='CAT')
axes[1].axhline(0.95, color='r', linestyle='--', linewidth=1.5, label='Threshold')
axes[1].set_xlabel('Items administered')
axes[1].set_ylabel('Reliability')
axes[1].set_title(f'Reliability Growth (  = {theta_test_sample[example_idx]:.2f})')
axes[1].legend(fontsize=7)
axes[1].grid(True, alpha=0.3)

# Histogram
axes[2].hist(random_items, bins=15, alpha=0.6, label='Random', color='#1f77b4')
axes[2].hist(cat_items, bins=15, alpha=0.6, label='CAT', color='#2ca02c')
axes[2].set_xlabel('Number of items')
axes[2].set_ylabel('Frequency')
axes[2].set_title('Distribution of Test Lengths')
axes[2].legend()
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

efficiency_gain = (np.mean(random_items) - np.mean(cat_items)) / np.mean(random_items)
 ↪  * 100
print(f"Random: {np.mean(random_items):.1f} ± {np.std(random_items):.1f} items")
print(f"CAT: {np.mean(cat_items):.1f} ± {np.std(cat_items):.1f} items")
print(f"Efficiency gain: {efficiency_gain:.1f}% fewer items with CAT")
```

6.2.4 Stopping Rules and Practical Considerations

CAT requires a stopping criterion. Common choices include:

1. **Reliability threshold**: Stop when $R = 1 - \sigma^2_{\text{post}}/\sigma^2_{\text{prior}} \geq 0.95$
2. **Standard error threshold**: Stop when $\text{SE}(\hat{\theta}) \leq 0.3$
3. **Fixed length**: Administer exactly K items
4. **Information threshold**: Stop when additional items provide negligible information

For AI evaluation, practical constraints interact with statistical criteria:

- **Cost**: Each API call has a monetary cost; the stopping rule should account for evaluation budgets.
- **Time**: Evaluations must complete within deadlines.
- **Contamination**: Administering too many items from the same pool risks benchmark leakage into training data.

> **CAT FOR AI EVALUATION**
>
> Traditional CAT assumes deterministic responses: a human test-taker gives the same answer if asked the same question twice. AI models may or may not satisfy this depending on temperature and sampling settings.
>
> For deterministic evaluation (temperature = 0), CAT applies directly. For stochastic evaluation, we may need multiple samples per item, or methods that account for response variability.
>
> CAT also requires pre-calibrated item parameters. In a cold-start scenario (new benchmark), we must first collect data on a pilot sample of models before CAT can be deployed. This connects to the cold-start prediction problem addressed in Section 8.1.

6.2.5 D-Optimal Design

CAT is *sequential* optimal design—it selects one item at a time. But sometimes we need to design a fixed test: selecting K items from a pool of M candidates to administer to all models at once. This is the classical problem of *optimal experimental design*.

For a K-dimensional factor model with ability vector $U_i \in \mathbb{R}^K$, the Fisher information matrix from administering a set of items $\mathcal{J}$ is:

$$\mathcal{I}(\theta; \mathcal{J}) = \sum_{j \in \mathcal{J}} P_j(\theta)(1 - P_j(\theta))\, V_j V_j^\top$$

where $V_j \in \mathbb{R}^K$ is the factor loading vector for item j. Different optimality criteria lead to different item selection strategies:

- **D-optimal**: Maximize $\det(\mathcal{I})$—the volume of the confidence ellipsoid. This minimizes the generalized variance of the ability estimates.

- **A-optimal**: Minimize $\text{tr}(\mathcal{I}^{-1})$—the average variance of individual ability components.
- **E-optimal**: Maximize $\lambda_{\min}(\mathcal{I})$—the smallest eigenvalue. This ensures no ability dimension is poorly estimated.

```
#| label: d-optimal-design
#| autorun: true
#| fig-cap: "D-optimal item selection chooses items with diverse difficulties covering the target ability range, yielding substantially higher total information than random selection."

from scipy.optimize import minimize as sp_minimize

def compute_test_information(item_indices, beta_pool, theta_grid):
    """Compute total Fisher information of a test across an ability grid."""
    total_info = np.zeros_like(theta_grid)
    for j in item_indices:
        P = sigmoid(theta_grid - beta_pool[j])
        total_info += P * (1 - P)
    return total_info

def d_optimal_greedy(beta_pool, K, theta_grid):
    """Greedy D-optimal item selection for the Rasch model.

    Selects K items from the pool to maximize total information
    integrated over the theta grid (approximating D-optimality
    for the unidimensional case).
    """
    available = list(range(len(beta_pool)))
    selected = []

    for _ in range(K):
        best_item = None
        best_score = -np.inf

        for j in available:
            candidate = selected + [j]
            info = compute_test_information(candidate, beta_pool, theta_grid)
            # D-optimal criterion: product of information values
            # (log-determinant in 1D = log of information)
            score = np.sum(np.log(info + 1e-10))
            if score > best_score:
                best_score = score
                best_item = j

        selected.append(best_item)
        available.remove(best_item)

    return selected

```

```
# Large item pool
np.random.seed(42)
M_pool = 200
beta_pool = np.random.normal(0, 2, M_pool)
K = 20  # Select 20 items

theta_grid = np.linspace(-3, 3, 100)

# D-optimal selection
d_optimal_items = d_optimal_greedy(beta_pool, K, theta_grid)

# Random selection (multiple draws for comparison)
random_infos = []
for trial in range(50):
    random_items = np.random.choice(M_pool, K, replace=False)
    info = compute_test_information(random_items, beta_pool, theta_grid)
    random_infos.append(info)

d_opt_info = compute_test_information(d_optimal_items, beta_pool, theta_grid)
random_mean = np.mean(random_infos, axis=0)
random_std = np.std(random_infos, axis=0)

# Visualization
fig, axes = plt.subplots(1, 3, figsize=(6, 2))

# Test information functions
axes[0].plot(theta_grid, d_opt_info, 'g-', linewidth=2, label='D-optimal')
axes[0].plot(theta_grid, random_mean, 'b-', linewidth=2, label='Random (mean)')
axes[0].fill_between(theta_grid, random_mean - random_std, random_mean + random_std,
                     alpha=0.2, color='blue')
axes[0].set_xlabel('Ability ( )')
axes[0].set_ylabel('Test Information')
axes[0].set_title('Test Information Function')
axes[0].legend(fontsize=7)
axes[0].grid(True, alpha=0.3)

# Selected item difficulties
axes[1].hist(beta_pool[d_optimal_items], bins=12, alpha=0.7, color='green',
             label='D-optimal', density=True)
axes[1].hist(beta_pool, bins=30, alpha=0.3, color='gray',
             label='Full pool', density=True)
axes[1].set_xlabel('Item Difficulty ( )')
axes[1].set_ylabel('Density')
axes[1].set_title('Selected Item Difficulties')
axes[1].legend(fontsize=7)
axes[1].grid(True, alpha=0.3)

# Integrated information comparison
d_opt_total = np.trapz(d_opt_info, theta_grid)
```

```
random_totals = [np.trapz(ri, theta_grid) for ri in random_infos]

axes[2].hist(random_totals, bins=15, alpha=0.6, color='blue', label='Random')
axes[2].axvline(d_opt_total, color='green', linewidth=2, linestyle='--',
                label=f'D-optimal ({d_opt_total:.1f})')
axes[2].set_xlabel('Integrated Information')
axes[2].set_ylabel('Frequency')
axes[2].set_title('Total Information Comparison')
axes[2].legend(fontsize=7)
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"D-optimal integrated info: {d_opt_total:.1f}")
print(f"Random integrated info: {np.mean(random_totals):.1f} ±
 ↪  {np.std(random_totals):.1f}")
print(f"D-optimal advantage: {(d_opt_total / np.mean(random_totals) - 1) * 100:.1f}%
 ↪  more information")
```

D-optimal design produces item pools with difficulties spread across the target ability range, ensuring high information everywhere. Random selection tends to cluster items near the center of the pool's difficulty distribution, leaving the tails poorly covered.

6.2.6 Design for Paired Comparisons

When evaluation is based on paired comparisons (as in the Chatbot Arena), the design problem takes a different form. Instead of selecting items, we must decide which pairs of models to compare. Under the Bradley-Terry model, the information from comparing models i and k about their strength difference $\theta_i - \theta_k$ is:

$$I_{ik}(\theta) = P_{ik}(1 - P_{ik}), \quad P_{ik} = \sigma(\theta_i - \theta_k)$$

This is maximized when the models are evenly matched ($P_{ik} = 0.5$, i.e., $\theta_i = \theta_k$). The design problem is to choose a tournament schedule—which pairs to compare, and how often—that maximizes the precision of the estimated ratings.

Classical solutions include **balanced incomplete block designs** (BIBDs), where each pair of models is compared equally often. For AI evaluation arenas, adaptive matchmaking algorithms serve the same role as CAT: they select matchups that are most informative given current rating estimates. This is precisely what the Chatbot Arena does when it pairs models with similar Elo ratings.

6.3 Discussion Questions

1. **Adaptive testing for AI**. What are the practical challenges in deploying CAT for AI evaluation? Consider: determinism of model responses, cost of API calls, benchmark contamination, and the need for pre-calibrated items.
2. **Optimal vs. fixed design**. When is it better to use adaptive testing (CAT) versus a fixed D-optimal test? What factors determine this choice in practice?
3. **Paired comparison design**. The Chatbot Arena uses adaptive matchmaking to pair models with similar Elo ratings. What are the advantages and disadvantages of this approach compared to a balanced tournament design?
4. **Evaluation budgets**. A team has a fixed budget of 10,000 API calls to evaluate 50 models on a benchmark with 500 items. Design an evaluation strategy that maximizes measurement precision under this constraint.

6.4 Bibliographic Notes

6.4.1 Optimal Experimental Design

The theory of optimal experimental design originates with Kiefer (1959). D-optimal design is covered in Atkinson et al. (2007). For the connection between Fisher information and test design in IRT, see Linden (2006) and Chang and Ying (2009). Wright and Stone (1979) provides an accessible introduction to test design under the Rasch model.

6.4.2 Computerized Adaptive Testing

CAT has a rich history beginning with Lord (1970). The Fisher information criterion for item selection was developed by Birnbaum (1968). For multidimensional CAT, see Segall (1996). Applications to AI evaluation are emerging; see Polo et al. (2024) for recent work on efficient benchmark design. S. Truong, Tu, et al. (2025) demonstrate the practical impact of adaptive testing in the scaling law setting: by combining IRT-calibrated item parameters with Elo-based adaptive item selection, they achieve comparable decision accuracy to full-scale evaluation using only 50 questions per benchmark — a 99.9% reduction in the query budget.

6.5 Exercises

6.5.1 Theoretical Exercises

Exercise 3.1 ($\star$): Show that Fisher information $I_j(\theta) = P_j(\theta)(1 - P_j(\theta))$ is maximized when $\theta = \beta_j$, and that the maximum value is $1/4$.

Exercise 3.2 ($\star\star$): For the 2PL model $P_j(\theta) = \sigma(\beta_{j2}(\theta - \beta_{j1}))$, derive the Fisher information and show that it equals $\beta_{j2}^2 P_j(1 - P_j)$. How does discrimination β_{j2} affect optimal item selection?

Exercise 3.3 ($\star\star$): Design a stopping rule for CAT that balances measurement precision with evaluation cost. Assume each API call costs \$0.01 and the value of reducing standard error by one unit is \$1. Find the cost-optimal stopping point.

6.5.2 Computational Exercises

Exercise 3.4 ($\star\star$): Extend the D-optimal design simulation to a 2-dimensional factor model with $K = 2$. Implement item selection using $j^* = \arg\max_j \det(\mathcal{J} + I_j)$ where $I_j = P_j(1-P_j)V_jV_j^\top$. Compare the selected item loading vectors to random selection.

Exercise 3.5 ($\star\star$): Compare the convergence of CAT across models with different ability levels. Does CAT require more items for extreme abilities (very high or very low)? Why?

Exercise 3.6 ($\star\star$): Investigate the sensitivity of CAT to misspecification of item parameters. If the calibration sample differs systematically from the test population, how does CAT performance degrade? Simulate this scenario.

7 Discovery

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Distinguish** the *confirmatory* setting of Chapter 3 through Chapter 5 (a construct is posited and the question is whether items measure it) from the *exploratory* setting of this chapter (only a collection of items is given and the construct structure must be discovered).
2. **Determine** how many constructs an item set measures using parallel analysis, distinguishing genuine factors from the eigenvalues that sampling noise alone produces.
3. **Compute** and interpret the structure matrix of a fitted factor model, and explain why standardized loadings—not raw loadings—are the comparable fingerprint of an item.
4. **Cluster** items by their structure vectors to reveal latent subgroups, and diagnose benchmark heterogeneity from the resulting inter-cluster correlations.
5. **Evaluate** the generalization of a learned factor model under masking schemes (entry-wise, row holdout, column holdout), distinguishing interpolation from extrapolation to unseen models and items.
6. **Contrast** the common-cause (latent-variable) account of item correlations with the network account, in which correlations arise from direct associations among items rather than a shared latent factor.
7. **Specify** the Gaussian graphical model (partial-correlation network) and the Ising model as the continuous and binary network models, and relate each to its estimation procedure.
8. **Explain** why latent-variable and network structures can be observationally close, connecting this underdetermination to the nomological equivalence of Chapter 1.

Scope of This Chapter

The preceding chapters all begin from a *named construct*. Chapter 1 asks whether an instrument measures the construct it claims to; Chapter 3 builds models that assume a single ability θ and tests whether items realize it; Chapter 4 estimates those models; Chapter 5 quantifies how precisely they measure. Throughout, *the thing being measured is decided in advance*, and the work is to certify that the items deliver it.

This chapter takes the opposite starting point. We are handed a collection of items—a benchmark, or a merge of several—*without* a vetted account of what they measure, and we ask the data to tell us: **How many distinct things does this item set measure? Which items measure which? And is "one ability" even the right shape for the dependence among items, or are the items related directly, with no common cause behind them?** This is the *exploratory* or *discovery* mode of measurement, the analogue of exploratory (as opposed to confirmatory) factor analysis in psychometrics. The factor model of Section 7.2 supplies the first tool; network models supply the second.

> **Notation**
>
> This chapter's central object is the factor model (Section 7.2): $U_i \in \mathbb{R}^K$ (person factor vector), $V_j \in \mathbb{R}^K$ (item loading vector), Z_j (item intercept), and $\Sigma = \text{Cov}(U)$ (factor covariance). New beyond it are the **structure matrix** S, the per-item **structure vector** S_j, and the network quantities: the precision matrix $\Theta = \Sigma^{-1}$, partial correlations $\rho_{jk \cdot \text{rest}}$, and the Ising couplings ω_{jk}.

The instrument for most of this chapter is the **factor model**, developed in Section 7.2 below. Where the IRT models of Chapter 3 give each model a single ability θ_i and ask whether the items realize it, the factor model gives a *vector* and lets the data say how many axes there are (Section 7.1), which items load on which (Section 7.3), and how those items group into latent skills (Section 7.4). Its role as the multidimensional escape hatch was flagged in Section 3.5; here we put it to systematic exploratory use. The final section then asks whether a latent-axis picture is the right one at all, or whether the items are better described as a network with no common cause behind them (Section 7.6).

7.1 How Many Constructs? Dimensionality Assessment

The first and bluntest discovery question is *how many* distinct things an item set measures. In the confirmatory setting of Chapter 3 one assumes the answer is one—a single ability θ—and tests whether the items realize it; here we let the data decide the count. The primary tool is **eigenvalue analysis** of the item correlation matrix: a unidimensional set has one eigenvalue far larger than the rest, while k genuine constructs produce k large eigenvalues. But raw eigenvalues mislead, because even pure noise produces a first eigenvalue above one (sampling fluctuations inflate the leading principal component of any finite matrix).

Parallel analysis (Horn 1965) corrects for this by comparing the observed eigenvalues against those from *random* data of the same dimensions, retaining a factor only when its eigenvalue exceeds the random-data threshold. The count k it returns is the number of factors we carry into the structure matrix of Section 7.3 and the clustering of Section 7.4—the rest of this chapter takes that count as given.

```
#| label: dimensionality-analysis
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(123)

# --- Simulate a 2-factor benchmark ---
n_models = 300
n_items = 20
n_factor1 = 10  # "verbal reasoning" items
n_factor2 = 10  # "quantitative reasoning" items
```

```
13
14  # Two latent factors with moderate correlation
15  factor_corr = 0.3
16  cov_matrix = np.array([[1.0, factor_corr], [factor_corr, 1.0]])
17  L = np.linalg.cholesky(cov_matrix)
18  factors = np.random.normal(0, 1, (n_models, 2)) @ L.T
19
20  # Factor loadings
21  loadings = np.zeros((n_items, 2))
22  loadings[:n_factor1, 0] = np.random.uniform(0.5, 0.9, n_factor1)  # Factor 1
23  loadings[n_factor1:, 1] = np.random.uniform(0.5, 0.9, n_factor2)  # Factor 2
24
25  # Generate continuous item scores (for eigenvalue analysis)
26  noise_var = 0.4
27  X = factors @ loadings.T + np.random.normal(0, np.sqrt(noise_var), (n_models,
     ↪  n_items))
28
29  # --- Eigenvalue analysis ---
30  corr_matrix = np.corrcoef(X.T)
31  eigenvalues = np.linalg.eigvalsh(corr_matrix)[::-1]  # Descending
32
33  # --- Parallel analysis (100 replications) ---
34  n_reps = 100
35  random_eigenvalues = np.zeros((n_reps, n_items))
36  for rep in range(n_reps):
37      X_random = np.random.normal(0, 1, (n_models, n_items))
38      corr_random = np.corrcoef(X_random.T)
39      random_eigenvalues[rep] = np.linalg.eigvalsh(corr_random)[::-1]
40
41  pa_threshold = np.percentile(random_eigenvalues, 95, axis=0)
42
43  # --- Visualization ---
44  fig, axes = plt.subplots(1, 2, figsize=(6, 2))
45
46  # Panel 1: Scree plot with parallel analysis
47  components = np.arange(1, n_items + 1)
48  axes[0].plot(components[:10], eigenvalues[:10], 'o-', color='#5B8DEE', linewidth=2,
49               markersize=8, label='Observed eigenvalues', zorder=3)
50  axes[0].plot(components[:10], pa_threshold[:10], 's--', color='#E8637A', linewidth=2,
51               markersize=6, label='95th percentile (random)', zorder=3)
52  axes[0].axhline(y=1, color='gray', linewidth=0.8, linestyle=':', label='Kaiser
     ↪  criterion')
53  # Shade retained factors
54  n_retained = sum(eigenvalues > pa_threshold)
55  for k in range(min(n_retained, 10)):
56      axes[0].axvspan(k + 0.6, k + 1.4, alpha=0.1, color='#45BF7C')
57  axes[0].set_xlabel('Component')
58  axes[0].set_ylabel('Eigenvalue')
59  axes[0].set_title('Scree Plot with Parallel Analysis')
```

```
axes[0].legend()
axes[0].set_xticks(components[:10])

# Panel 2: Loading structure
# Estimate loadings via PCA (first 2 components)
X_centered = X - X.mean(axis=0)
U, S, Vt = np.linalg.svd(X_centered, full_matrices=False)
pc_loadings = Vt[:2, :].T * S[:2] / np.sqrt(n_models - 1)

colors = ['#5B8DEE'] * n_factor1 + ['#E8637A'] * n_factor2
axes[1].scatter(pc_loadings[:, 0], pc_loadings[:, 1], c=colors, s=80,
                edgecolors='white', linewidth=0.5, zorder=3)
for i in range(n_items):
    axes[1].annotate(f'{i+1}', (pc_loadings[i, 0] + 0.02, pc_loadings[i, 1] + 0.02),
                     fontsize=7, color='gray')
axes[1].axhline(y=0, color='gray', linewidth=0.5)
axes[1].axvline(x=0, color='gray', linewidth=0.5)
axes[1].set_xlabel('PC1 Loading (Factor 1: Verbal)')
axes[1].set_ylabel('PC2 Loading (Factor 2: Quantitative)')
axes[1].set_title('Item Loading Structure')
from matplotlib.patches import Patch
axes[1].legend(handles=[
    Patch(facecolor='#5B8DEE', label='Verbal items (1-10)'),
    Patch(facecolor='#E8637A', label='Quantitative items (11-20)')
])

plt.tight_layout()
plt.show()

print(f"\nDimensionality Assessment:")
print(f"  First 5 eigenvalues: {', '.join(f'{e:.2f}' for e in eigenvalues[:5])}")
print(f"  Parallel analysis threshold: {', '.join(f'{e:.2f}' for e in
 ↪  pa_threshold[:5])}")
print(f"  Factors retained (parallel analysis): {n_retained}")
print(f"  Variance explained by 2 factors: {eigenvalues[:2].sum() /
 ↪  eigenvalues.sum():.1%}")
```

The scree plot (left) shows two eigenvalues clearly exceeding the parallel-analysis threshold (red dashed line), confirming two-dimensional structure. The loading plot (right) reveals the two-factor structure: verbal items (blue, 1–10) load on the first component, while quantitative items (red, 11–20) load on the second. A benchmark reporting a single "reasoning" score would conflate these two distinct abilities.

This count is the input to everything that follows. When parallel analysis returns $k = 1$, the unidimensional models of Chapter 3 are licensed and a single composite score is defensible; when $k > 1$, the honest response to a benchmark sold as one score is to report subscores per factor or revise the item set toward a single construct—and to fit a k-factor model (Section 7.2) whose loadings reveal *which* items measure *which* of the k constructs (Section 7.3).

7.2 The Factor Model

The instrument that turns a construct *count* into a *characterization* is the **factor model**. Where the IRT models of Chapter 3 assign each model a single ability θ_i, the factor model assigns a *vector*, letting each item load on more than one latent axis—exactly what a count $k > 1$ calls for.

Factor models provide an alternative perspective on latent variable measurement. While IRT focuses on item-level response probabilities, factor models focus on the covariance structure of responses. The classical linear factor model assumes observed variables are linear combinations of latent factors plus noise:

$$X_j = \lambda_{j1}F_1 + \lambda_{j2}F_2 + \cdots + \lambda_{jK}F_K + \epsilon_j$$

where:

- X_j is the observed score on item j
- F_k are latent factors (abilities, traits)
- λ_{jk} are factor loadings
- ϵ_j is item-specific error

In matrix notation: $X = \Lambda F + \epsilon$, where Λ is the $M \times K$ matrix of factor loadings. For binary data, we use a logistic link function:

$$P(Y_{ij} = 1|U_i, V_j, Z_j) = \sigma(U_i^\top V_j + Z_j)$$

where $U_i \in \mathbb{R}^K$ is the latent factor vector for person i, $V_j \in \mathbb{R}^K$ is the factor loading vector for item j, and $Z_j \in \mathbb{R}$ is the item intercept.

The key difference is the link function: the linear model uses an identity link ($X_j = ...$) for continuous responses, while the logistic model passes the linear predictor through the sigmoid $\sigma(\cdot)$ to produce a probability for binary responses.

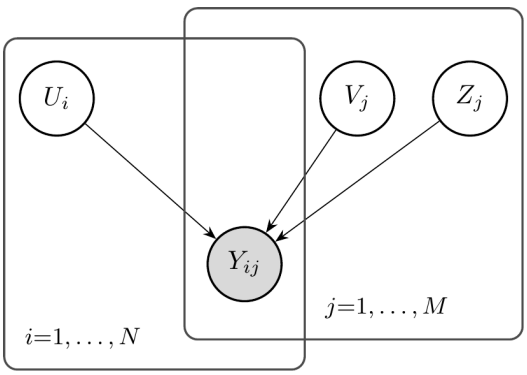


Figure 7.1
Plate diagram for the logistic factor model. The latent factor vector U_i interacts with item-specific loadings V_j and intercepts Z_j to produce responses.

When $K = 1$ and $V_j = 1$ for all j, the logistic factor model reduces to the Rasch model of Chapter 3: the multidimensional version simply lets each item carry its own loading *vector* rather than a fixed unit loading.

7.2.1 Fitting the Factor Model

We estimate the parameters (U, V, Z) by minimizing the binary cross-entropy between the predicted probabilities and the observed responses—the same maximum-likelihood objective developed for the Rasch model in Chapter 4, now over matrix-valued parameters. A compact implementation:

```
import torch
import torch.nn as nn
from torch.optim import LBFGS
import torch.nn.functional as F

class LogisticFM(nn.Module):
    """Logistic Factor Model for binary response data."""
    def __init__(self, N, M, K):
        super().__init__()
        self.U = nn.Parameter(torch.randn(N, K))  # Model abilities
        self.V = nn.Parameter(torch.randn(M, K))  # Item loadings
        self.Z = nn.Parameter(torch.randn(M, 1))  # Item intercepts

    def forward(self):
        return torch.sigmoid(self.U @ self.V.T + self.Z.T)
```

Interpretation

- U_i: latent ability vector of model i (position in K-dimensional capability space)
- V_j: latent property vector of item j (which capabilities the item measures)
- Z_j: overall item difficulty (independent of capability dimensions)
- σ: sigmoid function ensuring probabilities in $[0, 1]$

We train by minimizing binary cross-entropy loss with L-BFGS:

```
# Training setup
N, M = Y.shape
K = 2  # Number of latent dimensions
model = LogisticFM(N, M, K)

opt = LBFGS(
    model.parameters(),
    lr=0.1,
    max_iter=20,
    history_size=10,
```

```
    line_search_fn="strong_wolfe"
)

def closure():
    opt.zero_grad()
    probs = model()
    loss = F.binary_cross_entropy(probs[train_mask], Y[train_mask].float())
    loss.backward()
    return loss

# Training loop
for iteration in range(20):
    loss = opt.step(closure)
```

The model learns to decompose the response matrix into latent factors that capture the underlying structure of model capabilities and item characteristics. With a fitted model in hand, the next sections read its parameters—the loadings V_j via the structure matrix, and their groupings via clustering—and Section 7.5 asks how well those parameters predict held-out models and items.

7.3 Reading What Each Item Measures: The Structure Matrix

After fitting a multidimensional factor model, we obtain estimated item loadings $\hat{V}_j$ that describe how each item relates to the latent factors. However, raw factor loadings require standardization for interpretable comparisons across items and benchmarks.

The **structure matrix** S captures the correlation between each item's latent response and each factor:

Definition: Structure Matrix

For a logistic factor model, the latent response Y^*_{ij} can be written as $Y^*_{ij} = U_i^\top V_j + Z_j + \epsilon_{ij}$, where ϵ_{ij} follows a logistic distribution with variance $\pi^2/3$. The structure matrix entry S_{jk} is the correlation between item j's latent response and factor k:

$$S_{jk} = \text{Cor}(Y^*_{ij}, U_{ik}) = \frac{(V_j^\top \Sigma)_k}{\sqrt{\Sigma_{kk}}\sqrt{V_j^\top \Sigma V_j + \pi^2/3}}$$

where $\Sigma = \text{Cov}(U)$ is the factor covariance matrix.

The formula for S_{jk} is just a Pearson correlation, built from the standard recipe

$$\text{Cor}(A, B) = \frac{\text{Cov}(A, B)}{\text{SD}(A) \cdot \text{SD}(B)},$$

applied to $A = Y^*_{ij}$ and $B = U_{ik}$. Reading the three pieces in turn:

- **Numerator — the covariance**. Of the three terms in $Y^*_{ij} = U_i^\top V_j + Z_j + \epsilon_{ij}$, only $U_i^\top V_j$ depends on U; the item difficulty Z_j and logistic noise ϵ_{ij} are independent of U and drop out. This leaves

$$\mathrm{Cov}(Y^*_{ij}, U_{ik}) = \mathrm{Cov}(U_i^\top V_j,\, U_{ik}) = (V_j^\top \Sigma)_k.$$

- **Denominator — the two standard deviations**. The factor has $\mathrm{SD}(U_{ik}) = \sqrt{\Sigma_{kk}}$, and the latent response has

$$\mathrm{SD}(Y^*_{ij}) = \sqrt{\underbrace{V_j^\top \Sigma V_j}_{\text{from } U_i^\top V_j} + \underbrace{\pi^2/3}_{\text{from } \epsilon_{ij}}},$$

  since Z_j is a constant offset and the logistic error contributes variance $\pi^2/3$.

- **Why standardize**. The resulting $S_{jk} \in [-1, 1]$ is directly comparable across items, unlike the raw loadings V_j, which depend on the arbitrary scale of the factors.

A detailed derivation is given in Chapter 4.

We define the **structure vector** for item j as $S_j = (S_{j1}, S_{j2}, \ldots, S_{jK})^\top \in \mathbb{R}^K$, where each entry S_{jk} is given by the formula above. Stacking these row vectors yields the $M \times K$ **structure matrix** S, whose j-th row is $S_j^\top$. The structure vector S_j serves as a standardized fingerprint of item j: it tells us how strongly the item's latent response correlates with each factor, on a $[-1, 1]$ scale that is comparable across items.

The structure matrix has several important properties:

1. **Bounded values**: Each entry $S_{jk} \in [-1, 1]$, making comparisons intuitive.
2. **Interpretability**: High positive values indicate the item strongly measures that factor; negative values indicate inverse relationships.
3. **Clustering**: Items with similar structure vectors measure similar constructs and can be grouped together.

For AI benchmarks, the structure matrix reveals which questions tap into which capabilities. Two questions may both be "correct/incorrect" but load on different factors—one measuring reasoning, another measuring factual recall. This has important implications for how we interpret aggregate benchmark scores.

7.4 Item Clustering and Benchmark Heterogeneity

With the structure matrix in hand, we can cluster items into latent subgroups using standard clustering algorithms such as Gaussian Mixture Models (GMM). Each cluster represents a group of items sharing similar factor loadings—analogous to "skills" or "themes" within the benchmark. This approach is analogous to exploratory factor analysis in psychometrics, revealing whether

benchmarks are essentially unidimensional or composed of multiple, potentially antagonistic, latent skills.

> **! Benchmark Heterogeneity**
>
> A key insight from factor analysis applied to AI benchmarks is that **benchmarks are rarely homogeneous**. Intentionally or not, they often combine items that test different capabilities, and even a single benchmark item may test a combination of capabilities.
>
> Two models with identical mean scores may excel on different capability dimensions. For example, one model might be strong in reasoning but weak in factual recall, while another may have the reverse profile. When item clusters show weak or negative correlations with each other, the benchmark-level mean score becomes neither informative nor accurate about subgroup performance.

Within each benchmark, we can quantify inter-construct correlations by:

1. **Clustering items** based on their structure vectors using GMM with BIC for model selection
2. **Computing cluster means** for each model (average accuracy on items in each cluster)
3. **Correlating cluster means** across models to assess construct overlap

Strongly positive inter-cluster correlations indicate overlapping constructs, while weak or negative correlations suggest distinct and possibly conflicting capabilities being aggregated by the benchmark's mean score. This multidimensional pattern explains why two models with identical overall accuracies may excel on entirely different skill axes.

Factor models assign a feature vector to each item (the structure vector), allowing items to be clustered via standard algorithms. This helps interpret evaluation results that would otherwise be obscured by aggregate scores.

7.5 Generalization of the Factor Model

Fitting the factor model (Section 7.2) yields learned parameters (U, V, Z), and the previous two sections read structure out of them. But a discovered structure is only useful if it *generalizes*: do the learned parameters predict responses the model never trained on? To evaluate the robustness and transferability of learned factor models, we train and test them under various **masking schemes**, each representing a different notion of generalization. These masks determine which parts of the response matrix Y are visible during training and which are held out for evaluation.

7.5.1 Masking Schemes for Evaluation

Masking Type	Train Set	Test Set	Purpose
Entry-wise random	80% random entries	20% random entries	Interpolation under missing-at-random
Row holdout (random)	80% of models, all items	20% of models, all items	Generalization to unseen models
Row holdout (shifted)	Slice of models (small→large)	Disjoint slice	Covariate-shift generalization
Column holdout (random)	All models, 80% of items	All models, 20% of items	Generalization to unseen items
Column holdout (shifted)	Subset of benchmarks	Held-out benchmarks	Cross-domain transfer
Row-column block (L-mask)	$R_{tr} \times C_{tr}$	$R_{te} \times C_{te}$	Compositional generalization
Temporal split	Models before cutoff	Models after cutoff	Temporal generalization

These settings parallel psychometric validation tests where new examinees, items, or contexts probe the invariance of latent constructs.

7.5.2 *Implementation of Masking Functions*

```
import torch

def random_mask(data_idtor, pct=0.8):
    """Entry-wise random masking."""
    train_idtor = torch.bernoulli(data_idtor * pct).int()
    test_idtor = data_idtor.int() - train_idtor
    return train_idtor, test_idtor

def model_mask(data_idtor, pct_models=0.8, exposure_rate=0.3):
    """Row holdout: hold out unseen models."""
    train_row_mask = torch.bernoulli(torch.ones(data_idtor.shape[0]) *
    ↪ pct_models).bool()
    train_idtor = torch.zeros_like(data_idtor).int()
    train_idtor[train_row_mask, :] = data_idtor[train_row_mask, :]
    train_idtor[~train_row_mask, :], _ = random_mask(data_idtor[~train_row_mask, :],
    ↪ pct=exposure_rate)
    test_idtor = data_idtor - train_idtor
    return train_idtor, test_idtor

def item_mask(data_idtor, pct_items=0.8, exposure_rate=0.3):
    """Column holdout: hold out unseen items."""
```

```
    train_col_mask = torch.bernoulli(torch.ones(data_idtor.shape[1]) *
↪ pct_items).bool()
    train_idtor = torch.zeros_like(data_idtor).int()
    train_idtor[:, train_col_mask] = data_idtor[:, train_col_mask]
    train_idtor[:, ~train_col_mask], _ = random_mask(data_idtor[:, ~train_col_mask],
↪ pct=exposure_rate)
    test_idtor = data_idtor - train_idtor
    return train_idtor, test_idtor

def L_mask(data_idtor, pct_models=0.8, pct_items=0.8):
    """Row-column block (L-mask): compositional generalization."""
    train_row_mask = torch.bernoulli(torch.ones(data_idtor.shape[0]) *
↪ pct_models).bool()
    train_col_mask = torch.bernoulli(torch.ones(data_idtor.shape[1]) *
↪ pct_items).bool()
    train_idtor = torch.zeros_like(data_idtor).int()
    train_idtor[train_row_mask][:, train_col_mask] = data_idtor[train_row_mask][:,
↪ train_col_mask]
    test_idtor = data_idtor - train_idtor
    test_idtor[train_row_mask, :] = 0
    test_idtor[:, train_col_mask] = 0
    return train_idtor, test_idtor
```

7.5.3 *Two-Stage Training for Holdout Generalization*

To avoid data contamination in row and column holdout experiments, we use a **two-stage training procedure**:

7.5.3.1 *Row Holdout: Estimating Parameters for Unseen Models*

When testing generalization to unseen models, we:

1. **Stage 1**: Train on known models to learn item parameters (V, Z)
2. **Stage 2**: Freeze (V, Z) and estimate ability parameters U for held-out models using their limited exposed responses

This ensures item parameters are learned without information from test models.

```
# Stage 1: Train on known models
test_row = test_idtor.max(axis=1).values  # Identify held-out models
model_stage1 = train_model(Y[~test_row, :], mask=train_idtor[~test_row, :])

# Freeze V, Z from Stage 1
V_frozen = model_stage1.V.detach()
Z_frozen = model_stage1.Z.detach()
```

```

# Stage 2: Estimate U for unseen models with frozen item parameters
model_stage2 = train_model(Y[test_row, :], mask=train_idtor[test_row, :],
                           V_fixed=V_frozen, Z_fixed=Z_frozen)
```

7.5.3.2 *Column Holdout: Estimating Parameters for Unseen Items*

When testing generalization to unseen items, we:

1. **Stage 1**: Train on known items to learn model parameters U
2. **Stage 2**: Freeze U and estimate item parameters (V, Z) for held-out items

```
# Stage 1: Train on known items
test_col = test_idtor.max(axis=0).values  # Identify held-out items
model_stage1 = train_model(Y[:, ~test_col], mask=train_idtor[:, ~test_col])

# Freeze U from Stage 1
U_frozen = model_stage1.U.detach()

# Stage 2: Estimate V, Z for unseen items with frozen model parameters
model_stage2 = train_model(Y[:, test_col], mask=train_idtor[:, test_col],
                           U_fixed=U_frozen)
```

Why Two-Stage Training?

The two-stage procedure prevents information leakage:

- **Row holdout**: Item parameters learned from training models should not contain information about test models
- **Column holdout**: Model parameters learned from training items should not contain information about test items

This mirrors the real-world scenario where we want to evaluate new models on pre-calibrated items, or calibrate new items using established models.

7.5.4 *Evaluation Across Masking Schemes*

For each masking scheme, we compute AUC on the held-out entries:

```
from torchmetrics import AUROC

masking_schemes = {
    "entry_random": random_mask,
    "row_holdout": model_mask,
```

```
    "col_holdout": item_mask,
    "L_mask": L_mask,
}

results = {}
auroc = AUROC(task="binary")

for name, mask_fn in masking_schemes.items():
    train_mask, test_mask = mask_fn(data_idtor)

    # Train model (with two-stage for row/col holdout)
    model = train_with_appropriate_stages(Y, train_mask, test_mask, name)

    # Evaluate on held-out entries
    P_hat = model().detach()
    auc = auroc(P_hat[test_mask.bool()], Y[test_mask.bool()])
    results[name] = auc.item()
    print(f"{name}: AUC = {auc:.3f}")
```

The factor model typically achieves AUC of 92–97% on random masking across benchmarks, demonstrating strong predictive power. Performance on row and column holdout tests the model's ability to generalize to new models and new items, respectively.

Application: Item Response Scaling Laws

The separability of model ability from item difficulty — the core property of IRT — has a powerful application to scaling laws. S. Truong, Tu, et al. (2025) show that by embedding IRT within the scaling law framework, one can factorize scaling law estimation from $O(M \times N)$ to $O(M + N)$, where M is the number of models (or checkpoints) and N is the number of questions.

Their key finding is that the IRT ability parameter θ scales linearly with the logarithm of pre-training compute: $\theta \approx a \cdot \log(\text{FLOP}) + b$. Combined with calibrated item parameters, this yields per-question scaling predictions: $\hat{R}_{ij}(x) = \sigma(d_j(\theta_i(x) - z_j))$. Because item parameters transfer across benchmarks that share the same measurement objective, ability estimated on one benchmark can predict performance on another — a direct validation of the cross-benchmark transfer tested in the masking experiments above.

In a study of 6,612 model checkpoints and 37,682 questions, this approach achieves comparable or superior decision accuracy to traditional scaling laws using only 50 questions per benchmark — a 99.9% reduction in queries. The approach uses Beta-IRT, which models empirical probability responses (token probabilities, pass rates) rather than binary correctness, capturing richer scaling signals.

7.6 When There Is No Common Cause: Network Models

The factor model, for all its flexibility, still rests on one structural commitment inherited from Chapter 1: a small set of latent variables is the *common cause* of every response, and items correlate *because* they share those causes. Hold the factors fixed and the items become independent—this is the local-independence assumption of Section 3.3, drawn as arrows fanning *out* from θ (or U)

to the responses. Discovery in the previous two sections stayed inside this picture: it asked only *how many* common causes there are and *which* items they touch.

Network psychometrics questions the picture itself. Suppose the items correlate not because a hidden ability radiates into all of them, but because they are *directly* coupled to one another—mastering the subskill behind item A is itself what makes item B solvable, with no third thing behind both. Then there is no construct to measure, only a web of mutual dependence to map. The object of discovery becomes the **graph** of direct associations: which items are connected, and how strongly, once every other item is accounted for.

The two accounts make a sharp, testable distinction at the level of *partial* (rather than marginal) association. Under a single latent factor, any two items are correlated marginally but become **conditionally independent given the factor**—the common cause explains away their association. Under a network, two coupled items stay associated *even after conditioning on every other item*, because their link is direct. So the discovery question is: conditioning on all other items, which associations survive?

7.6.1 The Gaussian graphical model

For continuous responses (or a latent-Gaussian surrogate for binary ones), the natural object is the **Gaussian graphical model** (GGM): a partial-correlation network. Let the items have covariance Σ and **precision matrix** $\Theta = \Sigma^{-1}$. The partial correlation between items j and k, controlling for all other items, is read directly off the precision matrix:

$$\rho_{jk\cdot\text{rest}} = -\frac{\Theta_{jk}}{\sqrt{\Theta_{jj}\,\Theta_{kk}}}.$$

The defining fact is that **a zero in the precision matrix is a conditional independence**: $\Theta_{jk} = 0 \iff$ items j and k are independent given all the others $\iff$ no edge j–k in the network. The marginal correlation matrix Σ is typically dense—everything correlates with everything through shared paths—but Θ can be sparse, and that sparsity *is* the network. Because a finite sample never yields exact zeros, one estimates a sparse $\hat{\Theta}$ directly with an ℓ_1 penalty (the **graphical lasso** (Friedman et al. 2008)), which sets small partial correlations to zero and returns an interpretable graph.

The contrast with the factor model is exact and worth stating as a slogan: the factor model explains a dense Σ by a *low-rank* structure ($V\Sigma V^\top$ plus diagonal noise); the network explains the same Σ by a *sparse inverse*. Low rank and sparse inverse are different hypotheses about *why* items covary, and the partial-correlation graph is what separates them.

7.6.2 The Ising model

For binary responses $Y \in \{0,1\}^M$—the usual benchmark case—the network analogue of the GGM is the **Ising model**, a pairwise Markov random field:

$$P(Y = y) = \frac{1}{Z(\tau, \omega)} \exp\Big(\sum_j \tau_j \, y_j + \sum_{j<k} \omega_{jk} \, y_j y_k \Big),$$

where τ_j is a threshold (an item's baseline propensity to be solved) and ω_{jk} is the **coupling** between items j and k—the direct association that remains after conditioning on every other item. As with the GGM, $\omega_{jk} = 0$ means no edge, and $Z(\tau, \omega)$ is the normalizing constant. Estimating the full joint is intractable for many items, but the conditional of each item given the rest is exactly a logistic regression on the other items,

$$P(Y_j = 1 \mid Y_{-j}) = \sigma\Big(\tau_j + \sum_{k \neq j} \omega_{jk} \, y_k \Big),$$

so the network can be recovered **node by node** with ℓ_1-penalized logistic regressions and the edge weights symmetrized—the *eLasso* / `IsingFit` procedure (Borkulo et al. 2014). This is the binary counterpart of the graphical lasso, and it is the model Exercise 4 of Chapter 3 pits against the Rasch model.

> **Definition: Nodewise neighborhood selection**
>
> For each item j, regress Y_j on all other items Y_{-j} with an ℓ_1 penalty; the nonzero coefficients identify j's **neighborhood** (its candidate edges). Doing this for every node and combining the results (an edge is kept if one or both endpoints select it) recovers the graph without ever forming the intractable joint likelihood. The same recipe estimates a GGM (linear regressions) or an Ising model (logistic regressions).

7.6.3 Networks and latent factors can look alike

It is tempting to read a sparse network as proof that "there is no latent ability." The inference is not that clean. Marginalizing a latent variable out of a local-independence model produces a *fully connected* network among the observed items—the common cause reappears as dense, uniform coupling (Marsman et al. 2018). Conversely, an Ising model whose coupling matrix has one dominant eigenvalue mimics a unidimensional latent factor: its summed score behaves almost exactly like a Rasch ability. The two stories therefore sit at opposite ends of a single family, and over a moderate sample they can be observationally close.

This is the same underdetermination we met in Chapter 1, where a correlated-factors model and a bifactor model reproduced the *same* observable covariance (Figure 1.2): the data fix the correlations, not the diagram behind them. Discovery, honestly practiced, returns a *menu* of structures compatible with the responses—one latent axis, several correlated axes, a sparse interaction network—and the choice among them is made by parsimony, out-of-sample prediction (Chapter 4), and substantive knowledge of the items, not by fit alone. What discovery *does* settle is the negative claim: when the partial-correlation graph refuses to collapse to a low-rank common cause, a single benchmark score is measuring a web, not a quantity.

The demonstration below makes the GGM-versus-correlation distinction concrete on the same Open LLM Leaderboard slice used in Section 3.5. The marginal correlations among items are dense; their partial correlations—the precision-matrix view—are far sparser, and that gap is exactly the structure a network model reads as *direct* item-to-item dependence.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: partial-correlation-network
#| autorun: true
#| fig-cap: "Marginal vs. partial item correlations on a 30-item slice of the Open LLM
 ↪  Leaderboard. Left: the marginal correlation matrix is dense---almost every pair of
 ↪  items is positively correlated, the signature of a shared latent ability. Right:
 ↪  the partial correlations (from the inverse covariance) are far sparser; most
 ↪  marginal association is explained away by conditioning on the other items, and
 ↪  only a few direct item-to-item links survive. Sparsity in the right panel, not
 ↪  density in the left, is what a network model treats as structure."

import json
import numpy as np
import matplotlib.pyplot as plt
from pyodide.http import open_url

raw = json.loads(open_url("data/openllm_subset.json").read())
Y = np.array(raw["data"], dtype=float)

# Keep the 30 items with the most balanced solve rates (informative, non-degenerate).
p = Y.mean(axis=0)
keep = np.argsort(np.abs(p - 0.5))[:30]
Yc = Y[:, keep]

```

```
# Marginal correlation matrix.
C = np.corrcoef(Yc, rowvar=False)

# Partial correlations from the (regularized) precision matrix.
S = np.cov(Yc, rowvar=False) + 1e-2 * np.eye(Yc.shape[1])  # ridge for invertibility
Theta = np.linalg.inv(S)
d = np.sqrt(np.diag(Theta))
P = -Theta / np.outer(d, d)        # partial correlations
np.fill_diagonal(P, 1.0)

fig, axes = plt.subplots(1, 2, figsize=(6.4, 3.0))
for ax, M, title in [(axes[0], C, "Marginal correlation"),
                     (axes[1], P, "Partial correlation")]:
    im = ax.imshow(M, vmin=-0.6, vmax=0.6, cmap="RdBu_r")
    ax.set_title(title, fontsize=9)
    ax.set_xticks([]); ax.set_yticks([])
fig.colorbar(im, ax=axes, fraction=0.046, pad=0.04)
plt.show()

frac = lambda M: np.mean(np.abs(M[np.triu_indices_from(M, 1)]) > 0.1)
print(f"off-diagonal |corr| > 0.1 : marginal = {frac(C):.2f}, partial =
 ↪  {frac(P):.2f}")
```

The marginal panel is the dense positive manifold a common-cause model predicts; the partial panel keeps only the associations that survive conditioning on everything else. When that residual graph is sparse, a low-rank latent model is the parsimonious explanation. When it stays dense and irreducible, the items are coupled directly—network-like—and no single ability orders them.

7.7 *Exercises*

Exercise 1 (*): A benchmark's items split into two GMM clusters whose model-level mean scores correlate at $r = -0.2$. Explain, in terms of the structure vectors S_j, what this negative correlation implies about the benchmark, and why reporting a single mean score is misleading here.

Exercise 2 (**): Show that under a single-factor model with local independence, any two items are conditionally independent given the factor, so their *partial* correlation controlling for the factor is zero even though their *marginal* correlation is positive. Relate this to the precision-matrix characterization of Section 7.6.1.

Exercise 3 (**): Implement nodewise neighborhood selection for the Ising model. Given a binary response matrix Y, fit an ℓ_1-penalized logistic regression of each item on the others, symmetrize the coefficient matrix, and plot the recovered network. Compare the edge set to the off-diagonal sparsity of the inverse covariance from a Gaussian graphical model on the same data.

Exercise 4 (***): Simulate data from (a) a unidimensional Rasch model and (b) an Ising model whose coupling matrix has one dominant eigenvalue. Fit both a one-factor model and a graphical lasso to each data set. Show empirically that the two generating mechanisms are hard to distinguish by fit alone, and discuss which auxiliary evidence (Section 7.6.3) would adjudicate.

Exercise 5 (**): Implement a cross-validation procedure for selecting between the Rasch, 2PL, and factor models (Section 7.2). Apply it to benchmark data with different numbers of items and models. When does the additional complexity of the 2PL or a multidimensional factor model improve out-of-sample prediction over the unidimensional Rasch model?

Discussion 1: Discovery returns a *menu* of structures compatible with the same responses (one latent axis, several correlated axes, a sparse network). For AI evaluation specifically, what substantive knowledge about *items* would justify preferring a network account over a latent-ability account, or vice versa? Does the answer change for a reasoning benchmark versus a knowledge benchmark?

Discussion 2: The unidimensional IRT estimation of Chapter 4 extends to the multidimensional factor model fit in Section 7.2, but new challenges arise—chief among them rotational indeterminacy (the loadings V are identified only up to an orthogonal rotation) and choosing the number of factors K. How do the learning procedures change relative to the single-ability case, and how does parallel analysis (Section 7.1) help resolve the choice of K?

7.8 Bibliographic Notes

The exploratory tradition this chapter draws on is exploratory factor analysis, whose modern treatment and the structure-versus-pattern-matrix distinction are covered in standard texts; the application to AI benchmark heterogeneity via the standardized structure matrix is developed in this book and its companion papers. Parallel analysis, the eigenvalue-against-noise procedure for choosing the number of factors, is due to Horn (1965) and remains the most reliable of the simple dimensionality heuristics, outperforming the Kaiser eigenvalue-greater-than-one rule it is plotted against in Section 7.1. Network psychometrics emerged in the 2010s as an alternative to the reflective latent-variable model: the Gaussian graphical model and the graphical lasso for sparse precision estimation are due to Friedman et al. (2008), and the Ising model with nodewise ℓ_1-logistic estimation (eLasso) to Borkulo et al. (2014). The observational closeness of Ising networks and latent-variable models is analyzed by Marsman et al. (2018), which formalizes the sense in which marginalizing a latent variable yields a network and a low-rank network mimics a latent factor—the discovery-side counterpart of the nomological equivalence discussed in Chapter 1.

8 PREDICTABILITY

8.1 Amortized Models

A **factor model** generalizes the IRT models of this chapter to several latent dimensions, assigning each model i an ability vector $U_i \in \mathbb{R}^K$ and each item j a loading vector $V_j \in \mathbb{R}^K$ and intercept Z_j, with $P(Y_{ij} = 1) = \sigma(U_i^\top V_j + Z_j)$; we develop it as an instrument for *discovering* latent structure in Section 7.2. Fit to a benchmark in the usual way, it learns a latent parameter vector for each person (U_i) and each item (V_j, Z_j) by optimizing over the observed response matrix. This works well when we have enough responses for every entity, but it has two limitations: (1) the number of parameters grows linearly with the number of persons and items, and (2) new persons or items that were not present during training have no learned representation. **Amortized models** address both limitations by replacing per-entity parameter optimization with a learned function that maps observable features to latent parameters.

In the standard factor model, each item j has its own learned parameters V_j and Z_j. An amortized model instead learns a function f_θ that predicts these parameters from item features:

$$[\hat{V}_j, \hat{Z}_j] = f_\theta(x_j)$$

where x_j is a feature vector for item j (e.g., a text embedding of the question). Similarly, person parameters can be amortized via a function g_ϕ:

$$\hat{U}_i = g_\phi(m_i)$$

where m_i is a feature vector for person i (e.g., model metadata such as parameter count, architecture family, and release date). The predicted correctness probability is then:

$$\hat{P}_{ij} = \sigma(\hat{U}_i^\top \hat{V}_j + \hat{Z}_j)$$

The key advantage is that f_θ and g_ϕ generalize to *new* items and persons not seen during training, as long as their features are available.

Amortized measurement models are closely related to **neural collaborative filtering** (NCF), a family of methods from recommendation systems that use neural networks to model user-item interactions. In the recommender systems literature, the response matrix Y records user preferences (ratings, clicks); in our setting, it records model correctness. Both problems share the same structure: predicting missing entries of a sparse matrix.

Classical matrix factorization decomposes the response matrix as $Y \approx UV^\top + Z$, which is exactly the logistic factor model from Section 7.2 without the sigmoid. NCF extends this by replacing the inner product $U_i^\top V_j$ with a neural network $h_\psi(U_i, V_j)$ that can learn non-linear interactions between person and item latent factors. In the measurement context, this corresponds to replacing the linear logit $U_i^\top V_j + Z_j$ with a more flexible function:

$$P(Y_{ij} = 1) = \sigma(h_\psi(U_i, V_j, Z_j))$$

This increased flexibility can capture complex interaction patterns—for instance, a model that is strong on reasoning but specifically weak on multi-step arithmetic—that a linear factor model would need many dimensions to approximate. The cost is reduced interpretability: the latent factors no longer have a clean geometric interpretation as "ability axes."

NCF AND AMORTIZED MODELS

Neural collaborative filtering and amortized models address complementary problems. NCF uses neural networks to model the *interaction* between latent factors (replacing $U_i^\top V_j$ with a learned function). Amortized models use neural networks to *predict* the latent factors from features (replacing per-entity U_i, V_j with functions of observable features). The two ideas can be combined: use amortized encoders to produce latent factors, then a neural interaction function to produce predictions.

9 Intervention

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Formulate** causal models for AI evaluation and distinguish causal from associational claims about benchmark performance.
2. **Apply** structural causal models (SCMs) to represent the data-generating process behind evaluation data, including the roles of training data, model architecture, and benchmark design.
3. **Identify** when distribution shift—between training, calibration, and deployment populations—threatens the validity of evaluation conclusions, and characterize shift types (covariate, label, concept).
4. **Use** interventional and counterfactual reasoning to diagnose construct-irrelevant variance: does the benchmark score change because ability changed, or because something else did?
5. **Explain** the connection between Borsboom's causal theory of validity and modern causal inference: a benchmark is valid if and only if the target construct causally produces variation in scores.
6. **Apply** doubly robust estimation to correct for selection bias in adaptive and non-representative evaluations, connecting importance weighting, model-based imputation, and prediction-powered inference.
7. **Evaluate** when benchmark results generalize across deployment contexts and when they do not, using transportability criteria and conformal inference.

Suggested Lecture Plan

This chapter can be covered in **3 lectures** (75-90 minutes each):

Lecture 1: Causal Models for Evaluation

- From correlation to causation in benchmark scores (20 min)
- Structural causal models and the evaluation DAG (25 min)
- Interventional reasoning: diagnosing contamination and CIV (20 min)
- Borsboom's causal validity revisited (10 min)

Lecture 2: Distribution Shift and Transportability

- Types of distribution shift: covariate, label, concept (20 min)
- When do benchmark results generalize? Transportability (25 min)
- The evaluation-as-bandit framing (20 min)
- Hands-on: shift visualization (10 min)

Lecture 3: Robust Estimation and Conformal Prediction

- Three estimators: DM, IPW, doubly robust (25 min)
- Connection to prediction-powered inference (15 min)

– Conformal prediction under distribution shift (20 min)
– Hands-on: DR estimation and weighted conformal (15 min)

Notation

This chapter introduces causal and distributional notation: $\text{do}(X = x)$ (intervention), $P^{(s)}/P^{(t)}$ (source/target distributions), π_0/π (logging/target policies), $w(x)$ (importance weights), $\hat{V}_{\text{DR}}$ (doubly robust estimator), and $C_\alpha(x)$ (conformal prediction sets). See the notation appendix[1] for the complete notation reference.

[1] notation.qmd

9.1 From Association to Causation in Benchmark Scores

Model A outperforms Model B on a popular reasoning benchmark. What can we conclude? The association is clear: $\mathbb{E}[Y \mid \text{Model} = A] > \mathbb{E}[Y \mid \text{Model} = B]$. But does this mean Model A has greater reasoning ability? Or did Model A's training data happen to include problems similar to the benchmark items, giving it a memorization advantage that has nothing to do with reasoning?

This distinction—between associational and causal claims—is the central concern of this chapter. In Chapter 3, we introduced Borsboom's realist definition of validity: a test is valid for measuring an attribute if and only if (a) the attribute exists and (b) variation in the attribute *causally produces* variation in test scores (Borsboom 2005). In Chapter 1, we developed the practical implications of this definition through content, criterion, construct, external, and consequential validity. This chapter provides the *formal causal machinery* to make these ideas precise.

9.1.1 The Causal Hierarchy

Pearl (2009) distinguishes three levels of causal reasoning, each requiring progressively stronger assumptions:

1. **Association** $P(Y \mid X)$: What does observing X tell us about Y? *Example:* "Models with more parameters tend to score higher on MMLU." This is a correlation that could arise from many causal structures.
2. **Intervention** $P(Y \mid \text{do}(X = x))$: What happens to Y if we *set* X to x, regardless of what would have occurred naturally? *Example:* "If we increase the context window from 4K to 128K tokens, does the reasoning score improve?" This requires knowing the causal structure, not just the joint distribution.
3. **Counterfactual** $P(Y_x \mid X = x')$: What *would have happened* to Y if X had been x, given that we actually observed $X = x'$? *Example:* "Would this model have scored lower on the benchmark if its training data had not included the benchmark's source documents?" Counterfactuals reason about individual cases, not populations.

Most benchmark analyses operate at Level 1. A leaderboard reports $P(Y \mid \text{Model})$—the association between model identity and score. But the claims we want to make are at Level 2 or 3: "This model *has better reasoning ability*" is a causal claim about what produces the score.

9.1.2 Structural Causal Models

A **Structural Causal Model** (SCM) provides the formal language for causal reasoning (Pearl 2009).

Definition: Structural Causal Model

An SCM is a tuple $\mathcal{M} = (U, V, F, P(U))$ where:

- U is a set of exogenous (background) variables
- $V = \{V_1, \ldots, V_n\}$ is a set of endogenous variables
- $F = \{f_1, \ldots, f_n\}$ is a set of structural equations, $V_i = f_i(\text{pa}(V_i), U_i)$, where $\text{pa}(V_i) \subseteq V \setminus \{V_i\}$
- $P(U)$ is a distribution over exogenous variables

Each SCM induces a directed acyclic graph (DAG) where edges point from $\text{pa}(V_i)$ to V_i. The DAG encodes conditional independence structure: $V_i \perp\!\!\!\perp V_j \mid S$ if S *d-separates* V_i and V_j in the DAG.

9.1.3 The Evaluation DAG

We now construct an SCM for the AI evaluation process. The key endogenous variables are:

- D: **Training data** (corpus composition, size, quality)
- A: **Architecture** (transformer variant, parameter count, training recipe)
- θ: **Latent ability** (the construct the benchmark claims to measure)
- B: **Benchmark design** (item pool, selection criteria, format)
- β: **Item properties** (difficulty, discrimination, content)
- F: **Formatting** (prompt template, few-shot examples, system prompt)
- Y: **Observed score** (the benchmark outcome)

The causal structure is:

$$D \to \theta, \quad A \to \theta, \quad \theta \to Y, \quad \beta \to Y, \quad B \to \beta, \quad F \to Y$$

When the benchmark is *valid* in Borsboom's sense, the primary path from D and A to Y passes through θ: training data and architecture determine ability, and ability determines the score. The item properties β moderate this relationship (harder items produce lower scores for the same ability), but do not confound it.

Validity threats as DAG pathologies. The threats identified in Section 1.3 correspond to specific DAG structures:

- **Contamination** (Section 1.3.3): A direct path $D \to Y$ that bypasses θ. The model's training data includes benchmark items, so the score reflects memorization rather than ability. This is a *confounding* path that inflates the observed association between θ and Y.
- **Construct-irrelevant variance** (Section 1.3.1): A path $F \to Y$ that does not pass through θ. Prompt formatting affects the score independent of ability. This is an additional cause of Y that is not part of the target construct.
- **Differential item functioning** (Section 5.11.1): An interaction $A \to Y$ that is not mediated by θ. Certain architectures have advantages on certain items independent of their ability level.

! Validity = No Unblocked Backdoor Paths

Under Borsboom's causal framework, a benchmark is valid if and only if the *only* systematic path from the data-generating process to the observed score passes through the target construct θ. Every unblocked non-θ path is a validity threat. The diagnostic tools from Section 5.11 — DIF analysis, dimensionality assessment, item-fit statistics — are empirical tests for whether such paths exist.

9.2 Interventional and Counterfactual Diagnostics

The evaluation DAG from Section 9.1.3 allows us to reason about *interventions*—what happens when we deliberately change one variable while holding others fixed.

9.2.1 The Back-Door Adjustment

The back-door criterion (Pearl 2009) identifies when observational data suffices to estimate causal effects. A set of variables Z satisfies the back-door criterion relative to an ordered pair (X, Y) if: (i) no node in Z is a descendant of X, and (ii) Z blocks every path between X and Y that contains an arrow into X.

When the back-door criterion is satisfied:

$$P(Y \mid \text{do}(X = x)) = \sum_z P(Y \mid X = x, Z = z) P(Z = z) \tag{9.1}$$

This *adjustment formula* removes confounding by averaging over the confounder distribution. For AI evaluation, this means: to estimate the causal effect of ability θ on score Y, we must adjust for all confounders—variables that cause both θ and Y through non-θ paths.

9.2.2 Diagnosing Contamination

The contamination diagnostic from Section 1.3.3 can be formalized as an interventional question. Let D_{overlap} denote the subset of training data that overlaps with benchmark items. The causal question is:

$$P(Y \mid \text{do}(D_{\text{overlap}} = \emptyset)) \stackrel{?}{=} P(Y \mid D_{\text{overlap}} = \emptyset)$$

If these are equal, removing the overlapping data does not change the score *beyond* what we would expect from the reduced training set. If they differ, the overlap was providing a direct $D \to Y$ path (memorization) rather than contributing to θ (genuine learning).

In practice, we cannot perform this intervention on already-trained models. But we can approximate it through:

- **Chronological splits**: Items created after the training data cutoff cannot be memorized. Comparing scores on pre- vs. post-cutoff items estimates the contamination effect.
- **Canary detection**: Embedding unique strings in benchmark items and testing whether models reproduce them provides evidence for the $D \to Y$ path (Jacovi et al. 2023).
- **Item-fit analysis**: The infit/outfit statistics from Section 5.11.4 detect items where some models perform unexpectedly well—a signature of the memorization path.

9.2.3 Counterfactual Reasoning for DIF

Differential Item Functioning (Section 5.11.1) has a natural counterfactual interpretation. An item i exhibits DIF between groups $g = 0$ and $g = 1$ if:

$$P(Y_{i,g=1} = 1 \mid \theta) \neq P(Y_{i,g=0} = 1 \mid \theta)$$

This is a counterfactual statement: "For a model with ability θ, would the response to item i have been different if the model belonged to the other group, holding ability fixed?" The "group" variable g (e.g., model architecture family) has a *direct* effect on the item response that is not mediated by θ.

Under the evaluation DAG, DIF corresponds to a path $A \to Y$ that does not pass through θ—an architectural advantage on specific items that reflects something other than the target construct. The Mantel-Haenszel procedure from Section 5.11.2 estimates this direct effect by stratifying on θ (using total score as a proxy), which blocks the $A \to \theta \to Y$ path and isolates the $A \to Y$ direct effect.

! When Is a Score Difference Causal?

A score difference $Y_A - Y_B$ between two models reflects a genuine *ability* difference only if all paths from the data-generating process to Y that do not pass through θ are blocked. Common unblocked paths in AI evaluation:

- **Training data overlap** ($D \to Y$ bypassing θ): memorization inflates scores
- **Prompt formatting** ($F \to Y$): format preferences affect scores independent of ability
- **Evaluation order** (position bias in LLM-as-judge): systematic bias from presentation order

The causal diagnostics in this section and the statistical diagnostics in Section 5.11 are complementary approaches

to detecting these unblocked paths.

9.3 Distribution Shift and Transportability

In Section 1.2.4, we raised the question of whether benchmark results generalize beyond the specific conditions of the evaluation. This section provides the formal framework.

9.3.1 Setup

Let $P^{(s)}(X, Y)$ denote the **source** (benchmark) joint distribution over items X and responses Y, and $P^{(t)}(X, Y)$ the **target** (deployment) distribution. The fundamental question is: when does a performance estimate computed under $P^{(s)}$ remain valid under $P^{(t)}$?

The joint distribution factorizes as $P(X, Y) = P(Y \mid X)P(X)$, giving rise to a taxonomy of shift types based on which component changes.

DEFINITION: TYPES OF DISTRIBUTION SHIFT

1. **Covariate shift**: $P^{(s)}(X) \neq P^{(t)}(X)$ but $P^{(s)}(Y \mid X) = P^{(t)}(Y \mid X)$. The *item distribution* changes but the model's *conditional behavior* is stable. Example: a benchmark oversamples easy items; deployment sees a harder mix.
2. **Label shift** (prior probability shift): $P^{(s)}(Y) \neq P^{(t)}(Y)$ but $P^{(s)}(X \mid Y) = P^{(t)}(X \mid Y)$. The *base rate* of outcomes changes. Example: a benchmark has 50% correct base rate; deployment tasks have 10%.
3. **Concept drift**: $P^{(s)}(Y \mid X) \neq P^{(t)}(Y \mid X)$. The *relationship between items and responses* changes. Example: a coding benchmark becomes outdated as programming conventions evolve, so the correct answers shift.

9.3.2 Covariate Shift in AI Evaluation

Covariate shift is the most common and most correctable form of shift. In AI evaluation, it arises whenever the benchmark item pool is not representative of the deployment item distribution.

Consider a benchmark designed to measure mathematical reasoning. If the item pool is drawn disproportionately from algebra (easy) rather than from combinatorics, number theory, and analysis (hard), then the benchmark's item distribution $P^{(s)}(X)$ is shifted toward easy items relative to the deployment distribution $P^{(t)}(X)$. The model's behavior on any given item—its conditional response $P(Y \mid X)$—is the same in both settings. But the overall accuracy computed under $P^{(s)}$ overestimates deployment accuracy because easy items are overweighted.

This connects directly to the content validity discussion in Section 1.2.1: a benchmark with poor content validity (non-representative item sampling) is a benchmark with covariate shift relative to the construct domain.

9.3.3 Concept Drift in AI Evaluation

Concept drift is more dangerous because the model's conditional behavior itself changes. In AI evaluation, concept drift occurs when:

- **Model updates**: A model is updated via RLHF, fine-tuning, or system prompt changes. The IRT parameters calibrated on the old model no longer apply.
- **Prompt template changes**: Switching from a zero-shot to a few-shot template changes $P(Y \mid X)$ even for the same items.
- **Temporal evolution**: Programming languages evolve, scientific knowledge advances, and social norms shift. A benchmark's "correct" answers may become outdated.

Under concept drift, no reweighting of the source distribution can recover the target performance. The model must be re-evaluated on data drawn from the target distribution, or the concept drift must be modeled explicitly.

9.3.4 Transportability

When can we transport causal conclusions from one setting to another? Bareinboim and Pearl (2016) formalize this through **selection diagrams**: DAGs augmented with special "selection" nodes S that indicate which mechanisms differ between source and target.

The key result (stated informally): a causal effect is transportable from source to target if the differences between environments, as encoded by S, can be "adjusted away" using the causal structure. Specifically, the target-domain causal effect is identifiable from the source data plus knowledge of which mechanisms differ.

For AI evaluation, this means: a benchmark result is transportable to a new deployment context if we can identify *what changed* between the benchmark setting and deployment, and the evaluation DAG provides a path to adjust for those changes. If the only change is the item distribution (covariate shift), importance weighting suffices. If the model's behavior changes (concept drift), we need additional assumptions or new data from the target domain.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
```

```
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: shift-taxonomy
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit

np.random.seed(42)

fig, axes = plt.subplots(1, 3, figsize=(6, 2))

theta_range = np.linspace(-3, 3, 200)

# --- Panel 1: Covariate Shift ---
# Source: easy items (beta ~ N(-1, 0.5))
# Target: broad items (beta ~ N(0, 1.5))
beta_source = np.random.normal(-1, 0.5, 500)
beta_target = np.random.normal(0, 1.5, 500)

axes[0].hist(beta_source, bins=30, alpha=0.6, color='#5B8DEE', density=True,
 ↪  label='Source (benchmark)')
axes[0].hist(beta_target, bins=30, alpha=0.6, color='#E8637A', density=True,
 ↪  label='Target (deployment)')
axes[0].set_xlabel(r'Item difficulty ($\beta$)')
axes[0].set_ylabel('Density')
axes[0].set_title('Covariate Shift')
axes[0].legend()
axes[0].set_xlim(-4, 4)

# Inset: same ICC in both
ax_inset1 = axes[0].inset_axes([0.55, 0.55, 0.4, 0.35])
theta_ex = 0.5
p_source = expit(theta_ex - theta_range)
ax_inset1.plot(theta_range, p_source, '#5B8DEE', linewidth=1.5, label='Source')
ax_inset1.plot(theta_range, p_source, '#E8637A', linewidth=1.5, linestyle='--',
 ↪  label='Target')
ax_inset1.set_title('Same P(Y|X)')
ax_inset1.set_xticks([])
ax_inset1.set_yticks([])

# --- Panel 2: Label Shift ---
# Source: 50% correct base rate
# Target: 10% correct base rate
n_items = 1000
```

```
y_source = np.random.binomial(1, 0.5, n_items)
y_target = np.random.binomial(1, 0.1, n_items)

categories = ['Correct', 'Incorrect']
source_rates = [y_source.mean(), 1 - y_source.mean()]
target_rates = [y_target.mean(), 1 - y_target.mean()]

x_pos = np.arange(len(categories))
width = 0.35
axes[1].bar(x_pos - width/2, source_rates, width, color='#5B8DEE', alpha=0.7,
 ↪  label='Source')
axes[1].bar(x_pos + width/2, target_rates, width, color='#E8637A', alpha=0.7,
 ↪  label='Target')
axes[1].set_xticks(x_pos)
axes[1].set_xticklabels(categories)
axes[1].set_ylabel('Proportion')
axes[1].set_title('Label Shift')
axes[1].legend()
axes[1].set_ylim(0, 1.1)

# --- Panel 3: Concept Drift ---
# Source: Rasch with beta=0
# Target: Rasch with beta shifted AND discrimination changed
beta_fixed = 0.0
p_source_icc = expit(theta_range - beta_fixed)
p_target_icc = expit(1.5 * (theta_range - 0.5))  # Different discrimination and
 ↪  difficulty

axes[2].plot(theta_range, p_source_icc, '#5B8DEE', linewidth=2.5, label='Source:
 ↪  P(Y|X)')
axes[2].plot(theta_range, p_target_icc, '#E8637A', linewidth=2.5, linestyle='--',
 ↪  label='Target: P(Y|X)')
axes[2].fill_between(theta_range, p_source_icc, p_target_icc, alpha=0.15,
 ↪  color='#F0A35C')
axes[2].set_xlabel(r'Ability ($\theta$)')
axes[2].set_ylabel('P(Correct)')
axes[2].set_title('Concept Drift')
axes[2].legend()
axes[2].set_ylim(-0.05, 1.05)

plt.tight_layout()
plt.show()

print("Distribution Shift Summary:")
print(f"  Covariate shift: source mean difficulty = {beta_source.mean():.2f}, "
      f"target mean difficulty = {beta_target.mean():.2f}")
print(f"  Label shift: source base rate = {y_source.mean():.2f}, "
      f"target base rate = {y_target.mean():.2f}")
print(f"  Concept drift: P(Y|X) changes between source and target")
```

The three panels illustrate the shift taxonomy. **Covariate shift** (left): the benchmark item pool is concentrated at easy difficulties (blue), while deployment items span a broader range (red), but the item characteristic curve is the same in both settings. **Label shift** (center): the base rate of correct responses differs between benchmark (50%) and deployment (10%). **Concept drift** (right): the relationship between ability and response probability itself changes—the target ICC (dashed red) has different discrimination and difficulty than the source (blue), so no reweighting of items can correct the estimate.

9.4 Off-Policy Evaluation and Doubly Robust Estimation

The distribution shift framework tells us *when* benchmark results are biased. This section develops the tools to *correct* that bias, drawing on the off-policy evaluation literature from contextual bandits.

9.4.1 The Evaluation-as-Bandit Framing

We reframe AI evaluation as an off-policy estimation problem. The key mapping is:

Bandit concept	AI evaluation translation
Context x	Evaluation item (content, format, difficulty)
Action a	Model's response
Reward r	Correctness score (0 or 1, or a continuous quality rating)
Logging policy $\pi_0(a \mid x)$	The benchmark design process—how items were selected and administered
Target policy $\pi(a \mid x)$	The model's behavior under deployment conditions

Definition: Off-Policy Evaluation

Given logged data $\{(x_t, a_t, r_t)\}_{t=1}^{n}$ collected under logging policy π_0, the off-policy evaluation problem is to estimate the **value** of a target policy π:

$$V(\pi) = \mathbb{E}_{x \sim P, a \sim \pi(\cdot|x)}[r(x, a)]$$

without deploying π to collect new data.

Why does this framing matter? Whenever items are selected non-uniformly—by adaptive testing (Section 6.2.2), by convenience sampling, or by any design process that makes some items more likely to appear than others—we are observing rewards under π_0 but want to estimate performance under π. Naive accuracy (the simple average of rewards) is biased because $\pi_0 \neq \pi$.

9.4.2 Three Estimators

Three classical estimators address this problem, each with different bias-variance tradeoffs.

The Direct Method (DM). Build a reward model $\hat{r}(x, a)$ and use it to predict rewards under the target policy:

$$\hat{V}_{\text{DM}} = \frac{1}{n} \sum_{t=1}^{n} \hat{r}(x_t, \pi) \tag{9.2}$$

where $\hat{r}(x_t, \pi) = \sum_a \pi(a \mid x_t) \hat{r}(x_t, a)$ integrates the reward model over the target policy's action distribution. In the AIMS context, $\hat{r}$ is precisely the IRT model from Chapter 3: $\hat{r}(x, a) = P(Y = 1 \mid \theta, \beta_x)$. The direct method is what we have been doing throughout the earlier modeling chapters (Chapter 3, Chapter 4): using the fitted IRT model to predict performance.

The direct method is biased when the reward model is misspecified. If the IRT model is wrong—perhaps the true data-generating process has interactions the model does not capture—the predictions will be systematically off.

Inverse Propensity Weighting (IPW). Instead of modeling the reward, model the selection process and reweight observations:

$$\hat{V}_{\text{IPW}} = \frac{1}{n} \sum_{t=1}^{n} \frac{\pi(a_t \mid x_t)}{\pi_0(a_t \mid x_t)} r_t \tag{9.3}$$

Each observation is weighted by the ratio of how likely the target policy was to take that action versus how likely the logging policy was. IPW is unbiased when the propensities are known, but can have very high variance when π_0 and π differ substantially (the weights become large).

> **! The CAT Propensity Problem**
>
> In computerized adaptive testing (Section 6.2.2), items are selected to maximize Fisher information at the current ability estimate. This means the item selection probability $\pi_0(x_t \mid \hat{\theta}_t)$ is non-uniform: items near the model's estimated ability are heavily oversampled, while very easy and very hard items are undersampled.
>
> Naive accuracy on CAT-selected items is biased because the item pool is deliberately non-representative. IPW corrects this by upweighting items that were unlikely to be selected by the adaptive algorithm. The propensities π_0 are known from the CAT algorithm's selection rule (the item with maximum Fisher information at the current estimate is selected with probability 1 in a greedy CAT, or with probability proportional to information in a stochastic variant).

The Doubly Robust Estimator (DR). The doubly robust estimator (Dudı□k et al. 2011; Robins et al. 1994) combines both approaches:

$$\hat{V}_{\text{DR}} = \frac{1}{n} \sum_{t=1}^{n} \left[\hat{r}(x_t, \pi) + \frac{\pi(a_t \mid x_t)}{\pi_0(a_t \mid x_t)} (r_t - \hat{r}(x_t, a_t)) \right] \tag{9.4}$$

Definition: Doubly Robust Estimator

The DR estimator augments the direct method prediction $\hat{r}(x_t, \pi)$ with a propensity-weighted correction term $\frac{\pi}{\pi_0}(r_t - \hat{r})$. It is **doubly robust**: consistent if *either* the reward model $\hat{r}$ or the propensity model π_0 is correctly specified (but not necessarily both).

- When $\hat{r}$ is correct: the residual $r_t - \hat{r}(x_t, a_t)$ has zero expectation, so the correction vanishes and $\hat{V}_{\text{DR}} \approx \hat{V}_{\text{DM}}$.
- When $\hat{r}$ is wrong but π_0 is correct: the IPW term corrects the model's bias, yielding an unbiased estimate.

Proof of double robustness. We verify that $\mathbb{E}[\hat{V}_{\text{DR}}] = V(\pi)$ when either model is correct. Taking expectations:

$$\mathbb{E}[\hat{V}_{\text{DR}}] = \mathbb{E}[\hat{r}(x, \pi)] + \mathbb{E}\left[\frac{\pi(a \mid x)}{\pi_0(a \mid x)}(r - \hat{r}(x, a))\right]$$

For the second term, conditioning on x:

$$\mathbb{E}\left[\frac{\pi(a \mid x)}{\pi_0(a \mid x)}(r - \hat{r}(x, a)) \,\middle|\, x\right] = \sum_a \pi_0(a \mid x)\frac{\pi(a \mid x)}{\pi_0(a \mid x)}(\mathbb{E}[r \mid x, a] - \hat{r}(x, a))$$

$$= \sum_a \pi(a \mid x)(r^*(x, a) - \hat{r}(x, a))$$

where $r^*(x, a) = \mathbb{E}[r \mid x, a]$ is the true reward function. If $\hat{r} = r^*$, this is zero. If π_0 is correct (which it is by construction in the importance weighting), the full expectation becomes:

$$\mathbb{E}[\hat{V}_{\text{DR}}] = \mathbb{E}[\hat{r}(x, \pi)] + \mathbb{E}_x\left[\sum_a \pi(a \mid x)(r^*(x, a) - \hat{r}(x, a))\right] = \mathbb{E}_x\left[\sum_a \pi(a \mid x) r^*(x, a)\right] = V(\pi)$$

9.4.3 Connection to Prediction-Powered Inference

Prediction-Powered Inference (PPI) (Angelopoulos et al. 2023) is a recently developed framework that is structurally equivalent to doubly robust estimation. PPI uses a small labeled dataset $\mathcal{D}_n = \{(X_i, Y_i)\}_{i=1}^n$ (expensive: actual model evaluations) and a large unlabeled dataset $\mathcal{D}_N = \{(X_i, \hat{Y}_i)\}_{i=1}^N$ (cheap: model predictions or automated scores) to estimate a population quantity $\mu = \mathbb{E}[Y]$:

$$\hat{\mu}_{\text{PPI}} = \frac{1}{N}\sum_{i \in \mathcal{D}_N} \hat{Y}_i + \frac{1}{n}\sum_{i \in \mathcal{D}_n}(Y_i - \hat{Y}_i) \tag{9.5}$$

The first term is the direct method estimate using the cheap predictions. The second term is the bias correction: it estimates the systematic error in $\hat{Y}$ using the labeled subsample. This is exactly the DR structure: imputation + correction.

The connection to the cold-start pipeline in Section 8.1 is direct. The PPE (Prediction-Powered Evaluation) approach from Chapter 3 learns a mapping from model embeddings and metadata to IRT parameters, producing predictions $\hat{Y}_{ij}$ for unseen model-item pairs. PPI provides the debiasing step: use a small set of actual evaluations to correct the PPE predictions, yielding valid confidence intervals for model performance. The PPE predictions serve as the imputation model $\hat{r}$; the actual evaluations provide the labeled correction $Y - \hat{Y}$.

9.4.4 Importance Weighting for Covariate Shift

When the shift between benchmark and deployment is purely covariate shift ($P^{(s)}(Y \mid X) = P^{(t)}(Y \mid X)$, $P^{(s)}(X) \neq P^{(t)}(X)$), the importance-weighted estimator corrects the bias (Shimodaira 2000):

$$\hat{\mu}_{\text{IW}} = \frac{\sum_{i=1}^{n} w(x_i) Y_i}{\sum_{i=1}^{n} w(x_i)}, \quad w(x_i) = \frac{P^{(t)}(x_i)}{P^{(s)}(x_i)} \tag{9.6}$$

This is the self-normalized variant, which is more stable than the unnormalized version. The weights $w(x_i)$ upweight items that are underrepresented in the benchmark relative to deployment, and downweight overrepresented items.

Practical challenges:

- **Density ratio estimation**: The weights require knowing or estimating the density ratio $P^{(t)}/P^{(s)}$. Methods include logistic regression on a domain classifier, kernel mean matching, and moment matching (Sugiyama et al. 2012).
- **Weight clipping**: Extreme weights inflate variance. Clipping weights at a maximum value M (e.g., $w_i \leftarrow \min(w_i, M)$) trades bias for variance reduction.
- **Effective sample size**: The effective sample size $n_{\text{eff}} = (\sum w_i)^2 / \sum w_i^2$ measures how much information the reweighted sample contains. When $n_{\text{eff}} \ll n$, the importance-weighted estimate is unreliable.

AI example. A reasoning benchmark oversamples algebra items (60% algebra, 20% geometry, 20% combinatorics). The deployment distribution is uniform across topics. Importance weighting reweights: algebra items get weight $w = 1/3 \div 3/5 = 5/9$, while geometry and combinatorics items get weight $w = 1/3 \div 1/5 = 5/3$. The corrected accuracy better reflects deployment performance.

```
#| label: dr-estimation
#| autorun: true
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.special import expit

np.random.seed(123)

# --- Simulate CAT-like adaptive item selection ---
n_models = 50
n_items_pool = 200
n_selected = 30  # Items selected per model by CAT
n_reps = 500

# True model abilities
theta_true = np.random.normal(0, 1, n_models)
# Item difficulties spanning the range
beta_pool = np.linspace(-3, 3, n_items_pool)

# True value: average accuracy under uniform item selection
true_values = np.zeros(n_models)
for j in range(n_models):
    true_values[j] = expit(theta_true[j] - beta_pool).mean()

# Run repeated experiments
dm_estimates = np.zeros((n_reps, n_models))
ipw_estimates = np.zeros((n_reps, n_models))
dr_estimates = np.zeros((n_reps, n_models))
naive_estimates = np.zeros((n_reps, n_models))

for rep in range(n_reps):
    for j in range(n_models):
        # CAT item selection: select items near theta (high Fisher info)
        # Propensity: proportional to Fisher information
        p_items = expit(theta_true[j] - beta_pool)
        fisher_info = p_items * (1 - p_items)  # Rasch Fisher info
        propensity = fisher_info / fisher_info.sum()  # Normalized propensity

        # Select items (with replacement for simplicity)
        selected = np.random.choice(n_items_pool, n_selected, p=propensity,
 ↪  replace=False)
        beta_selected = beta_pool[selected]
        prop_selected = propensity[selected]

        # Generate responses
        p_correct = expit(theta_true[j] - beta_selected)
        responses = np.random.binomial(1, p_correct)

        # Uniform target propensity
        target_prop = 1.0 / n_items_pool

        # --- Naive estimate (simple average) ---
```

```
        naive_estimates[rep, j] = responses.mean()

        # --- Direct Method: use true IRT model ---
        # Slight misspecification: use noisy theta estimate
        theta_hat = theta_true[j] + np.random.normal(0, 0.3)
        dm_pred = expit(theta_hat - beta_pool).mean()
        dm_estimates[rep, j] = dm_pred

        # --- IPW ---
        weights = target_prop / prop_selected
        weights = weights / weights.sum()  # Self-normalize
        ipw_estimates[rep, j] = (weights * responses).sum()

        # --- Doubly Robust ---
        r_hat = expit(theta_hat - beta_selected)  # IRT prediction for selected items
        r_hat_full = expit(theta_hat - beta_pool).mean()  # DM estimate
        correction = (weights * (responses - r_hat)).sum()
        dr_estimates[rep, j] = r_hat_full + correction

# --- Compute bias and RMSE ---
def compute_stats(estimates, true_vals):
    bias = (estimates.mean(axis=0) - true_vals).mean()
    rmse = np.sqrt(((estimates.mean(axis=0) - true_vals)**2).mean())
    return bias, rmse

stats = {
    'Naive': compute_stats(naive_estimates, true_values),
    'DM (IRT)': compute_stats(dm_estimates, true_values),
    'IPW': compute_stats(ipw_estimates, true_values),
    'DR': compute_stats(dr_estimates, true_values)
}

# --- Visualization ---
fig, axes = plt.subplots(1, 2, figsize=(6, 2))

methods = list(stats.keys())
biases = [stats[m][0] for m in methods]
rmses = [stats[m][1] for m in methods]
colors = ['#B07CD8', '#5B8DEE', '#F0A35C', '#45BF7C']

# Panel 1: Bias
bars1 = axes[0].bar(methods, biases, color=colors, alpha=0.8, edgecolor='white')
axes[0].axhline(y=0, color='black', linewidth=0.8)
axes[0].set_ylabel('Bias')
axes[0].set_title('Estimation Bias (500 replications)')
for bar, val in zip(bars1, biases):
    axes[0].text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.001,
                 f'{val:.4f}', ha='center', fontweight='bold')

```

```
# Panel 2: RMSE
bars2 = axes[1].bar(methods, rmses, color=colors, alpha=0.8, edgecolor='white')
axes[1].set_ylabel('RMSE')
axes[1].set_title('Root Mean Squared Error (500 replications)')
for bar, val in zip(bars2, rmses):
    axes[1].text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.001,
                 f'{val:.4f}', ha='center', fontweight='bold')

plt.tight_layout()
plt.show()

print(f"\nOff-Policy Estimation Results ({n_reps} replications, {n_models} models):")
print(f"  {'Method':<12} {'Bias':>8} {'RMSE':>8}")
print(f"  {'-'*28}")
for m in methods:
    print(f"  {m:<12} {stats[m][0]:>8.4f} {stats[m][1]:>8.4f}")
```

The simulation demonstrates the off-policy estimation problem under CAT-like adaptive item selection. The **naive** estimator (simple average of responses on adaptively-selected items) is biased because the CAT algorithm oversamples items near the model's ability level. The **Direct Method** (IRT prediction with noisy ability estimate) has low bias when the model is approximately correct but is sensitive to misspecification. **IPW** corrects the selection bias but has higher variance due to extreme weights. The **Doubly Robust** estimator combines the strengths of both, achieving the lowest RMSE by using the IRT model for imputation and IPW for bias correction.

```
#| label: importance-weighting
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit
from scipy.stats import norm

np.random.seed(456)

# --- Simulate covariate shift: benchmark oversamples easy items ---
n_models = 100
n_source_items = 200
n_target_items = 200

theta = np.random.normal(0.5, 1, n_models)

# Source: easy-biased item pool (mean difficulty = -1)
beta_source = np.random.normal(-1, 0.7, n_source_items)
# Target: broad item pool (mean difficulty = 0)
beta_target = np.random.normal(0, 1.5, n_target_items)

```

```
# Compute true accuracy under each distribution
acc_source = np.array([expit(th - beta_source).mean() for th in theta])
acc_target = np.array([expit(th - beta_target).mean() for th in theta])

# Importance-weighted estimate using source data
# Weight each source item by p_target(beta) / p_source(beta)
weights = norm.pdf(beta_source, 0, 1.5) / norm.pdf(beta_source, -1, 0.7)
weights = weights / weights.sum() * len(weights)  # Normalize to sum to n

# Compute IW-corrected accuracy for each model
acc_iw = np.zeros(n_models)
for j in range(n_models):
    p_correct = expit(theta[j] - beta_source)
    acc_iw[j] = (weights * p_correct).mean()

# --- Visualization ---
fig, axes = plt.subplots(1, 2, figsize=(6, 2))

# Panel 1: Item distributions
beta_range = np.linspace(-4, 4, 200)
axes[0].plot(beta_range, norm.pdf(beta_range, -1, 0.7), '#5B8DEE', linewidth=2.5,
             label='Source (benchmark)')
axes[0].plot(beta_range, norm.pdf(beta_range, 0, 1.5), '#E8637A', linewidth=2.5,
             label='Target (deployment)')
axes[0].fill_between(beta_range, norm.pdf(beta_range, -1, 0.7), alpha=0.15,
 ↪  color='#5B8DEE')
axes[0].fill_between(beta_range, norm.pdf(beta_range, 0, 1.5), alpha=0.15,
 ↪  color='#E8637A')
axes[0].set_xlabel(r'Item difficulty ($\beta$)')
axes[0].set_ylabel('Density')
axes[0].set_title('Item Distributions: Source vs Target')
axes[0].legend()

# Panel 2: Accuracy estimates
# Sort by target accuracy for clearer visualization
sort_idx = np.argsort(acc_target)
model_rank = np.arange(n_models)

axes[1].scatter(model_rank, acc_source[sort_idx], color='#5B8DEE', s=20, alpha=0.6,
                label=f'Naive (source): mean={acc_source.mean():.3f}')
axes[1].scatter(model_rank, acc_iw[sort_idx], color='#45BF7C', s=20, alpha=0.6,
                label=f'IW-corrected: mean={acc_iw.mean():.3f}')
axes[1].scatter(model_rank, acc_target[sort_idx], color='#E8637A', s=20, alpha=0.6,
                label=f'True target: mean={acc_target.mean():.3f}')
axes[1].set_xlabel('Model rank (by true target accuracy)')
axes[1].set_ylabel('Estimated accuracy')
axes[1].set_title('Covariate Shift Correction')
axes[1].legend(loc='lower right')

```

```
plt.tight_layout()
plt.show()

print(f"\nCovariate Shift Correction:")
print(f"  Naive (source) mean accuracy: {acc_source.mean():.3f}")
print(f"  IW-corrected mean accuracy:   {acc_iw.mean():.3f}")
print(f"  True target mean accuracy:    {acc_target.mean():.3f}")
print(f"  Naive overestimation:         {acc_source.mean() - acc_target.mean():.3f}")
print(f"  IW residual bias:             {acc_iw.mean() - acc_target.mean():.3f}")
```

The left panel shows the source (benchmark) and target (deployment) item difficulty distributions. The benchmark oversamples easy items (blue, centered at $\beta = -1$), while deployment items span a broader range (red, centered at $\beta = 0$). The right panel shows the consequence: naive accuracy computed on benchmark items (blue) systematically overestimates deployment accuracy (red). Importance weighting (green) corrects the bias by upweighting hard items that are underrepresented in the benchmark, bringing the estimates close to the true deployment accuracy.

9.5 Conformal Prediction Under Distribution Shift

The off-policy estimators in Section 9.4 correct *point estimates* of model performance. But practitioners also need *uncertainty quantification*: not just "the model's deployment accuracy is 0.78" but "the model's deployment accuracy is in $[0.72, 0.84]$ with 90% confidence." Conformal prediction provides this guarantee with minimal distributional assumptions.

Definition: Conformal Prediction

Given a calibration set $\{(X_i, Y_i)\}_{i=1}^{n}$ and a new input X_{n+1}, a conformal prediction set $C_\alpha(X_{n+1})$ satisfies:

$$P(Y_{n+1} \in C_\alpha(X_{n+1})) \geq 1 - \alpha$$

under the assumption that the calibration and test data are **exchangeable** (informally: drawn from the same distribution in no particular order). No parametric assumptions on the data distribution are required.

9.5.1 Split Conformal Prediction

The split conformal algorithm (Vovk et al. 2005) is the simplest conformal method:

1. Fit a model $\hat{f}$ on a training set.
2. On a held-out calibration set $\{(X_i, Y_i)\}_{i=1}^{n}$, compute nonconformity scores $s_i = |Y_i - \hat{f}(X_i)|$.
3. Let $\hat{q}$ be the $\lceil (1-\alpha)(1+1/n) \rceil / n$ quantile of $\{s_1, \ldots, s_n\}$.
4. For a new input X_{n+1}, predict $C_\alpha(X_{n+1}) = [\hat{f}(X_{n+1}) - \hat{q}, \hat{f}(X_{n+1}) + \hat{q}]$.

The coverage guarantee $P(Y_{n+1} \in C_\alpha) \geq 1 - \alpha$ holds for *any* model $\hat{f}$, *any* data distribution, and *any* sample size—as long as the calibration and test data are exchangeable.

9.5.2 When Exchangeability Fails

Under distribution shift, exchangeability fails: the calibration data is drawn from $P^{(s)}$ while the test data comes from $P^{(t)}$. This causes the conformal prediction set to lose its coverage guarantee. Specifically, if the target distribution concentrates on regions where the model is less accurate (harder items), the calibration-based quantile $\hat{q}$ underestimates the nonconformity scores at test time, leading to **undercoverage**.

9.5.3 Weighted Conformal Prediction

Tibshirani et al. (2019) show that coverage can be restored under covariate shift by replacing the uniform quantile with an **importance-weighted quantile**. The key insight is the same as for importance weighting in Section 9.4.4: reweight the calibration scores to match the target distribution.

The algorithm modifies step 3: instead of the uniform quantile, compute the weighted quantile of $\{s_1, \ldots, s_n, +\infty\}$ with weights $\{w_1, \ldots, w_n, w_{n+1}\}$ where $w_i = P^{(t)}(X_i)/P^{(s)}(X_i)$ and $w_{n+1} = P^{(t)}(X_{n+1})/P^{(s)}(X_{n+1})$. This yields:

$$P^{(t)}(Y_{n+1} \in C_\alpha(X_{n+1})) \geq 1 - \alpha$$

The coverage guarantee now holds under the *target* distribution, provided the importance weights are correct. Barber et al. (2023) extend this beyond covariate shift to more general forms of distribution shift.

```
#| label: conformal-prediction
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit
from scipy.stats import norm

np.random.seed(789)

# --- Simulate: calibrate on easy items, test on harder items ---
n_cal = 200   # Calibration items (source)
n_test = 200  # Test items (target)
n_models = 50

# Source: easy items; Target: harder items
beta_cal = np.random.normal(-0.5, 0.8, n_cal)
beta_test = np.random.normal(1.0, 1.0, n_test)
```

```
18
19  # Test at multiple nominal coverage levels
20  alphas = np.arange(0.05, 0.55, 0.05)
21  coverage_naive = np.zeros(len(alphas))
22  coverage_weighted = np.zeros(len(alphas))
23
24  for a_idx, alpha in enumerate(alphas):
25      covers_naive = 0
26      covers_weighted = 0
27      n_total = 0
28
29      for j in range(n_models):
30          theta_j = np.random.normal(0, 1)
31
32          # True accuracy on calibration and test items
33          p_cal = expit(theta_j - beta_cal)
34          p_test = expit(theta_j - beta_test)
35
36          # Generate binary responses
37          y_cal = np.random.binomial(1, p_cal)
38          y_test = np.random.binomial(1, p_test)
39
40          # Model prediction: IRT probability
41          r_hat_cal = p_cal
42          r_hat_test = p_test
43
44          # Nonconformity scores on calibration set
45          scores_cal = np.abs(y_cal - r_hat_cal)
46
47          # --- Naive conformal: uniform quantile ---
48          q_level = int(np.ceil((1 - alpha) * (n_cal + 1))) - 1
49          q_level = min(q_level, n_cal - 1)
50          sorted_scores = np.sort(scores_cal)
51          q_naive = sorted_scores[q_level]
52
53          # --- Weighted conformal: importance-weighted quantile ---
54          weights = norm.pdf(beta_cal, 1.0, 1.0) / norm.pdf(beta_cal, -0.5, 0.8)
55          weights = weights / weights.sum()
56
57          # Weighted quantile
58          sorted_idx = np.argsort(scores_cal)
59          cum_weights = np.cumsum(weights[sorted_idx])
60          q_idx = np.searchsorted(cum_weights, 1 - alpha)
61          q_idx = min(q_idx, n_cal - 1)
62          q_weighted = scores_cal[sorted_idx[q_idx]]
63
64          # Check coverage on test items
65          for i in range(n_test):
66              score_test = np.abs(y_test[i] - r_hat_test[i])
```

```
            covers_naive += (score_test <= q_naive)
            covers_weighted += (score_test <= q_weighted)
            n_total += 1

    coverage_naive[a_idx] = covers_naive / n_total
    coverage_weighted[a_idx] = covers_weighted / n_total

# --- Visualization ---
fig, axes = plt.subplots(1, 2, figsize=(6, 2))

# Panel 1: Coverage vs nominal level
nominal = 1 - alphas
axes[0].plot(nominal, nominal, 'k--', linewidth=1, label='Ideal (y = x)')
axes[0].plot(nominal, coverage_naive, 'o-', color='#E8637A', linewidth=2,
 ↪  markersize=6,
             label='Naive conformal')
axes[0].plot(nominal, coverage_weighted, 's-', color='#45BF7C', linewidth=2,
 ↪  markersize=6,
             label='Weighted conformal')
axes[0].fill_between(nominal, nominal - 0.03, nominal + 0.03, alpha=0.1, color='gray',
                     label='±3% tolerance')
axes[0].set_xlabel('Nominal coverage (1 -  )')
axes[0].set_ylabel('Empirical coverage')
axes[0].set_title('Coverage Calibration Under Shift')
axes[0].legend()
axes[0].set_xlim(0.45, 1.0)
axes[0].set_ylim(0.45, 1.05)

# Panel 2: Distribution comparison
beta_range = np.linspace(-3, 4, 200)
axes[1].plot(beta_range, norm.pdf(beta_range, -0.5, 0.8), '#5B8DEE', linewidth=2.5,
             label='Calibration (easy)')
axes[1].plot(beta_range, norm.pdf(beta_range, 1.0, 1.0), '#E8637A', linewidth=2.5,
             label='Test (hard)')
axes[1].fill_between(beta_range, norm.pdf(beta_range, -0.5, 0.8), alpha=0.15,
 ↪  color='#5B8DEE')
axes[1].fill_between(beta_range, norm.pdf(beta_range, 1.0, 1.0), alpha=0.15,
 ↪  color='#E8637A')
axes[1].set_xlabel(r'Item difficulty ($\beta$)')
axes[1].set_ylabel('Density')
axes[1].set_title('Calibration vs Test Item Distributions')
axes[1].legend()

plt.tight_layout()
plt.show()

print(f"\nConformal Prediction Under Covariate Shift:")
print(f"  {'Nominal':>8} {'Naive':>8} {'Weighted':>8} {'Gap (naive)':>12}")
for i in range(len(alphas)):
```

```
    nom = 1 - alphas[i]
    print(f"  {nom:>8.2f} {coverage_naive[i]:>8.3f} {coverage_weighted[i]:>8.3f} "
          f"{coverage_naive[i] - nom:>+12.3f}")
```

The left panel shows coverage calibration: the diagonal represents perfect calibration (empirical coverage = nominal coverage). Naive conformal prediction (red) falls below the diagonal—it promises 90% coverage but delivers less, because the calibration items (easy) are not representative of the test items (hard). Weighted conformal prediction (green) restores coverage by reweighting the calibration scores to match the test distribution. The right panel shows the source of the problem: the calibration distribution (blue) is shifted left (easier items) relative to the test distribution (red).

9.6 Putting It All Together: A Causal Audit

We close with a worked example that composes all tools from this chapter. Consider "CodeReason," a hypothetical benchmark claiming to measure coding reasoning ability in language models.

Step 1: Draw the evaluation DAG (Section 9.1.3). We identify: Training Data D (code repositories, documentation), Architecture A (transformer variant), Latent Coding Ability θ, Benchmark Items β (coding problems), Prompt Template F (instruction format), and Observed Score Y. We suspect two validity-threatening paths: (i) $D \to Y$ bypassing θ (some models trained on CodeReason's source repository), and (ii) $F \to Y$ (the benchmark uses a specific instruction format that favors certain models).

Step 2: Check for distribution shift (Section 9.3). The benchmark items are predominantly Python function-completion tasks at medium difficulty. The deployment context requires debugging, multi-file reasoning, and code review across Python, JavaScript, and Rust. This is covariate shift: the item distribution $P^{(s)}(\beta)$ is narrower than $P^{(t)}(\beta)$. We estimate density ratios using a domain classifier trained to distinguish benchmark items from deployment items.

Step 3: Apply doubly robust estimation (Section 9.4.2). We combine two components: (a) An IRT model fitted on the benchmark data provides $\hat{r}(x, a)$—the direct method estimate of each model's accuracy on any item. (b) The density ratio weights from Step 2 provide the importance weights. The DR estimator corrects the IRT predictions using the importance-weighted residuals. The result: an accuracy estimate that accounts for both the item selection bias and any IRT misspecification.

Step 4: Construct prediction intervals (Section 9.5.3). Using the importance weights from Step 2, we apply weighted conformal prediction to the DR-corrected estimates. The result: "Model X's deployment accuracy is in [0.62, 0.71] with 90% coverage." The interval is wider than it would be under no shift, reflecting the additional uncertainty from extrapolating to a different item distribution.

Step 5: Diagnose contamination (Section 9.2.2). We compare model performance on items created before vs. after the training data cutoff (chronological split). Models with suspected contamination show a 15-point accuracy gap between pre- and post-cutoff items. We run item-fit analysis (Section 5.11.4) and find that the pre-cutoff items have inflated outfit statistics for these models. Conclusion: the $D \to Y$ path is active for these models. We flag them and report both contamination-adjusted and unadjusted rankings.

This five-step causal audit integrates the tools from this chapter with the diagnostic methods from Section 5.11, providing a systematic approach to evaluating whether benchmark results are trustworthy enough to inform deployment decisions.

9.7 Discussion Questions

1. A benchmark uses Computerized Adaptive Testing (Section 6.2.2) to select items. A critic argues that the resulting accuracy is "biased because models only answered items matched to their ability level." Using the off-policy evaluation framework, explain whether this criticism is valid and how the estimate could be corrected if needed.

2. Two benchmarks both claim to measure "reasoning." On Benchmark A, Model X outperforms Model Y. On Benchmark B, the ranking reverses. Draw a DAG that explains this reversal (hint: consider Simpson's paradox). What causal assumptions would make the Benchmark A ranking the correct one?

3. Prediction-powered inference assumes that the model predictions $\hat{Y}$ are informative about the true labels Y. When might this assumption fail in AI evaluation? What are the consequences for the DR estimator's consistency, and how could the failure be detected?

4. A company evaluates its model on an internal benchmark (source) and wants to predict performance on a customer-facing deployment (target). What information is needed to determine whether the benchmark results are transportable? Which assumptions are untestable from the source data alone?

5. Conformal prediction guarantees *marginal* coverage: $P(Y \in C_\alpha) \geq 1 - \alpha$ on average over the test distribution. Why is this weaker than *conditional* coverage ($P(Y \in C_\alpha \mid X = x) \geq 1 - \alpha$ for each x), and when does the distinction matter for AI evaluation?

9.8 Bibliographic Notes

The mathematical framework for causal reasoning used in this chapter is due to Pearl (2009), who developed structural causal models, the do-calculus, and the graphical criteria for identifiability. An accessible introduction is Peters et al. (2017). The transportability theory—when causal conclusions can be transferred across settings—is developed by Bareinboim and Pearl (2016).

The taxonomy of distribution shift draws on a large literature. Covariate shift and importance weighting were formalized by Shimodaira (2000). Sugiyama et al. (2012) provide a comprehensive treatment of density ratio estimation methods. Quiñonero-Candela et al. (2009) survey the broader landscape of dataset shift in machine learning.

Doubly robust estimation originated in the biostatistics literature with Robins et al. (1994), who introduced the augmented inverse probability weighted estimator. Dudı□k et al. (2011) adapted the DR framework to contextual bandits and off-policy evaluation, which is the formulation we use here. The connection between DR estimation and semiparametric efficiency theory is developed in the biostatistics literature; for machine learning applications, see the survey by Dudík, Erber, Langford, and Li (2014).

Prediction-Powered Inference was introduced by Angelopoulos et al. (2023), who showed how to construct valid confidence intervals by combining a small labeled dataset with a large set of model predictions. The structural connection between PPI and doubly robust estimation is noted in their paper and further developed in follow-up work on PPI++.

Conformal prediction was introduced by Vovk et al. (2005). Tibshirani et al. (2019) extended conformal methods to handle covariate shift via importance weighting, and Barber et al. (2023) further generalized the framework beyond exchangeability.

The connection between Borsboom's causal validity and structural causal models was developed in Chapter 3. The evaluation DAG in this chapter makes that connection explicit, showing how each validity threat from Section 1.3 corresponds to a specific causal pathway. Data attribution methods—influence functions, TracIn, and TRAK—provide complementary tools for tracing the causal effect of training data on model predictions, but are beyond the scope of this chapter; see Koh and Liang (2017) for the foundational work.

9.9 Exercises

Theoretical

1. Show that under covariate shift ($P^{(s)}(Y \mid X) = P^{(t)}(Y \mid X)$, $P^{(s)}(X) \neq P^{(t)}(X)$), the importance-weighted estimator $\hat{\mu}_{\text{IW}} = \frac{1}{n}\sum_{i=1}^{n} w(x_i) Y_i$ with $w(x_i) = P^{(t)}(x_i)/P^{(s)}(x_i)$ is unbiased for $\mathbb{E}_{P^{(t)}}[Y]$. Under what conditions on the weight distribution does the variance of $\hat{\mu}_{\text{IW}}$ diverge?

2. Prove the double robustness property: show that $\mathbb{E}[\hat{V}_{\text{DR}}] = V(\pi)$ (Equation 9.4) if either (a) the reward model satisfies $\hat{r}(x, a) = \mathbb{E}[r \mid x, a]$ for all (x, a), or (b) the propensity $\pi_0(a \mid x)$ is correctly specified. Where does the proof rely on the assumption that the logged data is collected under π_0?

3. In a CAT procedure that selects the item maximizing Fisher information at the current ability estimate $\hat{\theta}$, derive the selection propensity $\pi_0(x_t \mid \hat{\theta}_t)$ for a Rasch model. Show that this propensity is maximized for items with $\beta_i \approx \hat{\theta}_t$ and decays for items far from the

current estimate. What is the effective sample size n_{eff} of the resulting IPW estimator as a function of the item pool's difficulty distribution?

4. Using the evaluation DAG from Section 9.1.3, show that benchmark contamination (a direct path $D \to Y$ bypassing θ) is *not identifiable* from observational data alone without additional assumptions. What minimal intervention (e.g., a chronological split or a canary-based test) would identify the contamination effect? State the identifying assumptions precisely.

Computational

5. Implement importance weighting for a benchmark with covariate shift. Simulate a Rasch model with $P^{(s)}(\beta) = \mathcal{N}(-1, 1)$ (easy-biased source) and $P^{(t)}(\beta) = \mathcal{N}(0, 1.5)$ (broader target). For 200 models, compare naive and IW-corrected accuracy estimates. Vary the degree of shift (by changing the source mean from -2 to 0) and plot the naive bias as a function of shift magnitude.

6. Simulate a full CAT procedure for 200 models on a 500-item Rasch pool. For each model, run a 30-item adaptive test using the maximum-information selection rule from Section 6.2.2. Compute naive accuracy, IPW-corrected accuracy, and DR-corrected accuracy. Introduce IRT misspecification (e.g., the true model is 2PL but the reward model assumes Rasch) and show that DR remains robust while DM degrades.

7. Implement weighted conformal prediction for AI evaluation under covariate shift. Calibrate on items drawn from $\mathcal{N}(-0.5, 0.8)$ and test on items from $\mathcal{N}(1.0, 1.0)$. Plot empirical coverage vs. nominal level for both standard and weighted conformal across the range $\alpha \in [0.05, 0.50]$. How does the effective sample size of the weighted calibration set affect the coverage guarantee?

Discussion

8. Angelopoulos et al. (2023)'s PPI uses cheap model predictions to augment expensive human labels. In AI evaluation, the "expensive" labels are actual model runs and the "cheap" predictions are the PPE cold-start predictions from Section 8.1. Design a practical PPI pipeline for evaluating a new model on 10,000 items when only 500 actual evaluations can be afforded. What are the key design choices (which 500 items to evaluate? how to estimate the bias correction? how to construct confidence intervals)? How does the accuracy of the PPE predictions affect the width of the resulting intervals?

10 Ecosystem

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Map** the stakeholders of a modern AI benchmark—builders, evaluators/platforms, adopters, investors, and regulators—and identify the decision each one delegates to the metric and the incentive each has to distort it.
2. **Explain** why an expanding audience turns Goodhart's Law from an academic curiosity into a structural force, using the economics of *signaling* and the *market for lemons*.
3. **Distinguish** the two directions of gaming: builders who game the metric and evaluators/platforms who game the audience, and classify real episodes into a mechanism-based taxonomy.
4. **Connect** each gaming channel to a design lever from Chapter 11, and identify which channels remain open research problems.
5. **Articulate** the governance frontier: multi-principal evaluation, the role of open-source models, and how to organize a trustworthy third-party evaluation ecosystem.

Suggested Lecture Plan

This chapter can be covered in **2 lectures** (75–90 minutes each):

Lecture 1: The Ecosystem and Its Failures

- From scoreboard to target: the expanding audience (15 min)
- Benchmarks as signaling and the market for lemons (20 min)
- A taxonomy of gaming: builder-side and evaluator-side (35 min)

Lecture 2: Governance

- The gaming-channel / design-lever map (20 min)
- Multi-principal evaluation and the strategic evaluator (25 min)
- The role of open-source; organizing the community (25 min)

Chapter Roadmap

This chapter is the *diagnosis* that motivates the *design* of Chapter 11. We ask three questions in sequence: **who are the actors** (Section 10.2), **what game do they play** (Section 10.3), and **how should the system be governed** (Section 10.5). The technical machinery for re-engineering the game—randomization, distribution correction, holdout mechanisms, welfare-aligned metrics—is developed in Chapter 11; here we establish *why* that machinery is needed and *which* failures it must close.

10.1 From Scoreboard to Target

> "When a measure becomes a target, it ceases to be a good measure." — Charles Goodhart (1975)

A benchmark begins life as an *internal* instrument: a model developer's private scoreboard for tracking research progress. In that regime Goodhart's Law is mostly latent—the only party who can optimize against the metric is the same party who benefits from an honest reading of it. The structural shift of the past few years is that the *audience* of a benchmark has expanded far beyond the developer who builds the model.

Today a single leaderboard number is consumed by:

- **Builders**, who use it to allocate research effort and to claim state-of-the-art.
- **Adopters** (enterprises, agencies, application developers), who use it as a procurement and due-diligence signal in build-vs-buy and vendor-selection decisions.
- **Investors**, who read benchmark leadership as a proxy for a technical moat when pricing funding rounds.
- **Governments and regulators**, who increasingly reference capability and safety evaluations in policy. *(TODO: cite the specific provisions to lean on—e.g., the EU AI Act's general-purpose-AI obligations and codes of practice, and the relevant US framework. Verify current status as of the writing date; the policy landscape moves quickly.)*

Once any of these parties can move resources on the basis of a score, the developer's reputation, funding, or regulatory standing depends on that score—and the data-generating process is no longer fixed. This is exactly the setting in which Goodhart's adversarial variant dominates (Section 11.1). The remainder of this chapter takes the audience expansion as the root cause and traces its consequences.

> **! The chapter's thesis**
>
> A public benchmark is a **signaling instrument**. As its audience expands beyond builders, *both sides* of the market acquire an incentive to game it: builders game the metric, and benchmark providers game the audience. Restoring trust is therefore not only a statistical problem (Chapter 5, Chapter 1) but an **institutional design** problem—the subject of Chapter 11 and of this chapter's governance discussion.

10.2 The Actors and Their Incentives

We organize the ecosystem as a set of principals, each delegating a decision to the metric, each with a characteristic way of distorting it.

Table 10.1
Stakeholders of a modern benchmark and how each can distort it

Actor	Decision delegated to the metric	Incentive to distort	Primary failure mode
Builder / developer	Where to allocate training effort; what to claim	Reputation, funding, market share	Optimize to the test (Section 10.3.1)
Evaluator / platform	What to publish and how to rank	Relevance, traffic, revenue, own-model favoritism	Game the audience (Section 10.3.2)
Adopter	Build vs. buy; which vendor	Lower procurement risk cheaply	Over-trust a decoupled proxy
Investor	How to price a round	Back the apparent leader	Reward narrative over capability
Regulator	Whether a system is safe to deploy	Public safety, accountability	Incentivize *sandbagging* (Section 10.3.1)

The classical evaluation model—and the formal game of Section 11.3—contains only the first two rows, and assumes the evaluator is a *benevolent social planner*. The central move of this chapter is to **relax that assumption**: the lower three rows add principals on the demand side, and the second row, once the evaluator has its own commercial incentives, becomes strategic. This is the multi-principal setting we return to in Section 10.5.1.

10.2.1 Benchmarks as Signaling and the Market for Lemons

Why should adopters and investors—parties who never train a model—rely on benchmarks at all? Because a benchmark is a *certification technology* that reduces an information asymmetry. The buyer cannot directly observe a model's quality before deploying it; the seller can. This is precisely Akerlof (1970)'s market for lemons: absent a credible quality signal, buyers rationally discount all sellers toward the average, and high-quality producers are driven out. A trustworthy benchmark is the device that lets a high-quality model *separate* itself—the AI analogue of Spence (1973)'s job-market signaling.

This framing does real work in the chapter:

- It explains the adopter and investor rows of Table 10.1 without hand-waving: both are *receivers* in a signaling game, differing only in the decision they take.
- It makes the cost of gaming precise. Gaming works by making the *cheap* signal (a tuned benchmark score) imitate the *expensive* one (genuine capability). When gaming is cheap relative to real improvement, the separating equilibrium collapses back to pooling—and the benchmark stops carrying information for *all* downstream audiences, even though it may still be locally useful to guide a builder's own research.

- It connects directly to Chapter 11: the design levers there (randomization, correction, holdout, welfare-aligned metrics) are all attempts to *raise the cost of the cheap signal* until separation is restored.

```
#| label: signal-decoupling
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

def spearman(a, b):
    # Spearman rank correlation = Pearson correlation of the ranks.
    ra, rb = np.argsort(np.argsort(a)), np.argsort(np.argsort(b))
    return np.corrcoef(ra, rb)[0, 1]

np.random.seed(0)

# A simple illustration of the chapter's thesis:
# as "gaming" becomes cheaper, the benchmark decouples from true utility.
M = 200                                  # number of models
true_utility = np.random.normal(0, 1, M)  # what society actually cares about

gaming_strengths = np.linspace(0, 3.0, 25)  # how much a model can boost its score by gaming
noise = 0.3
corrs = []
for g in gaming_strengths:
    gaming_effort = np.random.uniform(0, 1, M)     # heterogeneous propensity to game
    bench = true_utility + g * gaming_effort + np.random.normal(0, noise, M)
    corrs.append(spearman(bench, true_utility))

fig, ax = plt.subplots(1, 1, figsize=(3.5, 2.6))
ax.plot(gaming_strengths, corrs, color='#5B8DEE', lw=2)
ax.axhline(1.0, color='#888888', ls=':', lw=1)
ax.set_xlabel('Gaming strength (cheapness of the proxy)')
ax.set_ylabel("Benchmark-utility rank corr.")
ax.set_title('Decoupling under gaming')
ax.set_ylim(0, 1.05)
plt.tight_layout()
plt.show()
```

Figure. A deliberately simple illustration of the thesis: each model's benchmark score is its true utility plus a gaming term plus noise. As gaming becomes cheaper (x-axis), the rank correlation between the benchmark and true utility decays—the metric *decouples* from what downstream audiences care about. *(TODO: replace with a proper separating-vs-pooling signaling figure to make the Spence connection explicit.)*

10.3 A Taxonomy of Gaming

The single most important organizing decision for this chapter is to classify gaming **by mechanism, not by anecdote**. Two episodes that look different in the press (a contaminated math benchmark, a leaderboard scandal) may exploit the same channel—or the same-looking episodes may exploit different channels that need different defenses. We split the taxonomy by *who is gaming whom*.

10.3.1 Builders Game the Metric

This is the direction modeled formally in Section 11.3. Each sub-type names a distinct exploitable channel:

1. **Contamination / memorization**. Training data includes (or overlaps) evaluation items, so the benchmark measures recall rather than capability. Concrete: the OpenLLM Leaderboard's eventual archival amid contamination and overfitting concerns (Aidar Myrzakhan 2024); clean evidence from holdout re-tests such as the GSM1k re-examination of grade-school arithmetic (H. Zhang et al. 2024); and detection methods such as black-box contamination proofs (Oren et al. 2023). *Channel: the test items are public text that leaks into pretraining.*
2. **Adaptive overfitting via repeated querying**. Even without seeing the test items, repeated submissions against a fixed holdout let a builder overfit the leaderboard. This is the failure the Ladder mechanism and reusable-holdout techniques were designed to bound (Blum and Hardt 2015). *Channel: information leaks through the score itself across rounds (Section 11.5.4).*
3. **Strategic submission and selection**. A builder deploys models tuned for the queries a platform tends to surface, or selectively reports favorable runs. *(TODO: cite the Llama-4 Chatbot Arena episode (Singh et al. 2025) and the Terminal-Bench case once the references are settled.) Channel: the builder learns and targets the evaluation distribution.*
4. **Strategic post-training**. Ranking-aware fine-tuning shifts a model up a leaderboard without broad capability gains. *(TODO: cite "Model Rankings under Strategic Post-Training," `strategicposttraining2026`—verify authors/venue.) Channel: optimization directed at the rank statistic rather than the construct.*
5. **Sandbagging (the inverse)**. When the audience is a *regulator*, the incentive flips: a developer (or a model) may *underperform* on a dangerous-capability evaluation to appear safe (Weij et al. 2024). This is why government-facing evaluations need different safeguards than capability leaderboards; see the frontier dangerous-capability evaluation program of Phuong et al. (2024) and the red-teaming methodology of Chapter 12. *Channel: strategic concealment rather than strategic exaggeration.*

10.3.2 Evaluators Game the Audience

This is the direction the formal game of Chapter 11 does **not** yet model, and it is the chapter's novel thread. Once a benchmark provider has commercial or reputational incentives, the

provider is itself a strategic player:

- **Selective disclosure / pay-to-play**. Privileged providers obtain more matches, private pre-testing, or the ability to retract unfavorable results—asymmetries documented in the Chatbot Arena analysis of Singh et al. (2025; see also Chiang et al. 2024 for the platform). *Channel: the evaluator controls what the audience sees.*
- **Conflicts of interest**. A platform that ranks models while also building or investing in one of them has an incentive to tilt the playing field.
- **Market structure**. Why is trustworthy third-party evaluation hard to sustain as a business? The economics of the eval market—who pays, and whether the payer wants an honest signal—explain the recurring failure of independent eval startups (Liao 2025). This is a *governance* observation: the party best positioned to certify quality is often the party with the least incentive to do so honestly.

> **Why this split matters**
>
> Every defense in Chapter 11 assumes a benevolent evaluator and targets *builder-side* gaming. The evaluator-side failures of Section 10.3.2 are therefore **largely unaddressed by current mechanism design**—they are an open frontier (Section 10.5). Naming them as a separate direction is what keeps this chapter an argument rather than a list.

10.4 The Gaming-Channel / Design-Lever Map

This table is the spine that ties diagnosis (this chapter) to design (Chapter 11). Each gaming channel from Section 10.3 is closed—or left open—by a specific lever. The unclosed cells *are* the chapter's research agenda.

Table 10.2
Mapping each gaming channel to its design lever. Bold rows are evaluator-side failures not yet covered by the formal game.

Gaming channel	Section	Design lever	Where developed
Contamination / memorization	Section 10.3.1	Randomize & refresh tasks; live benchmarks	Section 11.5.1; (White et al. 2025; Jain et al. 2024)
Adaptive overfitting (repeated querying)	Section 10.3.1	Noise-gated holdout; differential privacy	Section 11.4.2
Learning the evaluation distribution	Section 10.3.1	Distribution correction toward uniform	Section 11.5.4.1
Rank-targeted / metric gaming	Section 10.3.1	Welfare-aligned (counterfactual) metrics	Section 11.7
Sandbagging on safety evals	Section 10.3.1	Adversarial / red-team elicitation	Chapter 12

Gaming channel	Section	Design lever	Where developed
Selective disclosure, pay-to-play	Section 10.3.2	*(open)* audit, transparency, conflict rules	Section 10.5
Evaluator conflicts of interest	Section 10.3.2	*(open)* institutional separation	Section 10.5.1

A practical reading of Table 10.2: the builder-side rows are well-served by the mechanisms of Section 11.9, while the evaluator-side rows require *governance* rather than *mechanism design*—rules about who may run a benchmark, what they must disclose, and who audits them.

10.5 The Governance Frontier

The remaining sections promote the chapter's hardest questions to first-class topics. These are genuinely open; we mark them as such.

10.5.1 Multi-Principal Evaluation and the Strategic Evaluator

The formal game of Section 11.3 has one evaluator and one builder, with the evaluator acting as a benevolent social planner. The ecosystem of Table 10.1 has *many* principals with conflicting objectives, and an evaluator who may itself be strategic. Open questions:

- **What is the right model of an evaluator with its own utility?** If the evaluator maximizes traffic or favors an affiliated model, what mechanism keeps its published ranking informative? (This is Discussion Question 1 of Chapter 11, promoted here to a research direction.)
- **What is a model of adopter and government behavior?** Both are receivers in the signaling game of Section 10.2.1, but with different loss functions and different abilities to audit. A formal account would let us ask which guarantees a benchmark must offer each.
- **Who watches the watchers?** Independent audit, reproducible evaluation infrastructure (Biderman et al. 2024), and conflict-of-interest rules are the institutional analogues of the technical levers in Chapter 11.

10.5.2 The Role of Open-Source Models

Open weights change the governance calculus in both directions, and it is worth stating the tension plainly:

- **For trust**: open weights enable independent third-party auditing, reproducible evaluation, and inspection that closed APIs preclude.
- **Against trust**: open weights make contamination and targeted gaming *easier*, since the model can be fine-tuned directly against any public benchmark.

(TODO: take a position here, and connect to the public/private benchmark work—the public version supports open auditing while a private holdout preserves an incentive-compatible signal. Add the citation for that paper once it has a key; `truong2025strategic` covers the private-benchmark design.)

10.5.3 Organizing for Trustworthy Evaluation

A closing, deliberately normative section: how should the research community organize a future in which capable AI is evaluated in ways that serve all of its audiences, not just its builders? *(TODO: this is the "intelligence serves all" theme—write the vision to leave the reader with, and connect it back to the predictive-evaluation half of the book.)*

10.6 Discussion Questions

1. The market-for-lemons framing (Section 10.2.1) predicts that high-quality models exit when no credible signal exists. Has the *opposite* been observed—a credible benchmark causing a flight to quality? What made it credible?
2. Table 10.2 leaves the evaluator-side rows open. Sketch a mechanism (technical or institutional) that would make selective disclosure costly for a platform.
3. Sandbagging (Section 10.3.1) inverts the usual gaming direction. Does *any* of the design machinery in Chapter 11 help against strategic *under*-performance, or is it fundamentally a red-teaming problem (Chapter 12)?
4. Adopters and investors consume the same number but take different decisions. Should a benchmark report *different* statistics to different audiences, and does that reintroduce the garbling problem of Section 11.7?
5. Argue both sides of Section 10.5.2: does open-sourcing models make the evaluation ecosystem more or less trustworthy on net?
6. The Kaggle era had a benevolent evaluator who *owned the test set* and *controlled submission*. List the structural features of the LLM era that break this, and rank them by how much damage each does.

10.7 Bibliographic Notes

Goodhart's Law and its variants are treated formally in Section 11.1; the strategic-classification and performative-prediction foundations are Hardt et al. (2016) and Perdomo et al. (2020).

Signaling and information economics. The market-for-lemons and signaling framing is Akerlof (1970) and Spence (1973). The information-design view of disclosure is developed in Chapter 11 via Kamenica and Gentzkow (2011).

Gaming in the wild. Leaderboard-platform asymmetries are analyzed by Singh et al. (2025; platform background: Chiang et al. 2024); contamination evidence and detection by H. Zhang et al. (2024) and Oren et al. (2023); the OpenLLM Leaderboard's archival by Aidar Myrzakhan (2024). Contamination-resistant "live" benchmarks include White et al. (2025) and Jain et al. (2024). Sandbagging and dangerous-capability evaluation are Weij et al. (2024) and Phuong

et al. (2024). The construct-validity critique of general benchmarks is Raji et al. (2021). The economics of the eval market is discussed by Liao (2025).

Design levers for every channel in Table 10.2 are developed in Chapter 11. Reproducible evaluation infrastructure is Biderman et al. (2024).

10.8 Exercises

1. **(Easy)** Fill in Table 10.1 for a familiar benchmark: for each actor, name the concrete decision and the concrete distortion.
2. **(Easy)** Modify the decoupling simulation (Section 10.2.1) so that gaming effort is correlated with low true utility (the worst models game hardest). How does the decoupling curve change?
3. **(Medium)** Formalize a one-shot signaling game between a builder (high- or low-quality) and an adopter, where a benchmark score is a costly signal. Derive the condition on gaming cost under which a separating equilibrium exists, and relate it to the design levers of Chapter 11.
4. **(Medium)** Pick one bold (evaluator-side) row of Table 10.2 and write a half-page design proposal that would move it from *open* to *closed*.
5. **(Hard)** Extend the Stackelberg game of Section 11.3 to a *strategic* evaluator whose utility is a convex combination of social welfare and own-traffic. Characterize when the evaluator's optimal mechanism still incentivizes broad builder capability.

11 Design

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Explain** why Goodhart's Law is a formal game-theoretic phenomenon rooted in the gap between a *construct* $\mathcal{C}$ and any single *distribution* D that operationalizes it, and classify its four variants.
2. **Distinguish** the three objects an evaluator might keep private — the construct $\mathcal{C}$, the operationalizing distribution D, and the realized set S — and classify mechanisms by which object they hide.
3. **Formalize** AI evaluation as a Stackelberg game and prove that a *transparent* benchmark fails, while randomization (hiding the realized draw from D) aligns incentives up to the builder's belief error $\mathrm{TV}(\mu, \tilde{\pi}_E)$.
4. **Explain** how reusable holdouts and the Ladder algorithm hide the set S and bound information leakage, and why dynamic benchmarks that refresh S without hiding D remain exploitable.
5. **Analyze** the information-variance tradeoff that hiding D creates in repeated evaluation, and derive distribution correction (and the optimal size k^*) as the response.
6. **State** the construct-validity protocol — publish $\mathcal{C}$ and public proxies while keeping D, S private — and the three properties of a durable evaluator (validity, durability, faithful execution).
7. **Model** metric design as a principal-agent problem and apply positional representation and proportionality criteria to select representative benchmark subsets.

Suggested Lecture Plan

This chapter can be covered in **3 lectures** (75-90 minutes each):

Lecture 1: From Goodhart to "What Are We Hiding?"

- Goodhart's Law and its four variants (15 min)
- Construct, distribution, set: the object-first framing (20 min)
- The Stackelberg evaluation game; why a static benchmark fails (25 min)
- Hands-on: deterministic vs. randomized evaluation simulation (15 min)

Lecture 2: Hiding the Set and Hiding the Distribution

- Reusable holdouts, the Ladder algorithm, and dynamic benchmarks (20 min)
- One-shot alignment via randomization; the information-variance tradeoff (25 min)
- Distribution correction and the optimal evaluation size (20 min)
- Hands-on: Pareto frontier and correction simulations (10 min)

Lecture 3: Construct, Metrics, and Synthesis

- The construct-validity protocol: publishing $\mathcal{C}$ and proxies (15 min)
- Metric design as a principal-agent problem; reveal, conceal, garble (25 min)

– Representative benchmark selection via social choice theory (20 min)
– Three properties of a durable evaluator; design principles (15 min)

Notation

This chapter introduces game-theoretic notation: F (task universe), F_E/F_M (evaluator/builder task sets), π_E/π_M (sampling distributions), $f(\theta)$ (task performance), u_E (evaluator utility), μ (social-relevance distribution), Δ_t (misalignment), ρ (correction rate), and k (tasks sampled per round). It also introduces the object-first vocabulary of privacy in evaluation: the construct $\mathcal{C}$, an operationalizing distribution D (formally the evaluator's sampling distribution π_E), a realized evaluation set S, the construct's full operationalization μ (the social-relevance weighting; Uniform(F) as a special case), and public proxy distributions D', D''. See the notation appendix[1] for the complete notation reference.

[1] notation.qmd

11.1 When Measurement Becomes a Target

> "When a measure becomes a target, it ceases to be a good measure." — Charles Goodhart (1975)

In previous chapters, we treated evaluation as a *statistical* problem: how to estimate latent abilities from noisy observations (Chapter 3), how to quantify measurement precision (Chapter 5), and how to assess whether a benchmark measures what it claims to measure (Chapter 1, Chapter 9). Throughout, we implicitly assumed that the data-generating process is *fixed* — that the act of measurement does not change the phenomenon being measured.

This assumption breaks down once benchmarks become *influential*. When a developer's reputation, funding, or regulatory standing depends on benchmark scores, the developer has every incentive to optimize specifically for the benchmark — not for the broader capability it claims to measure. The data-generating process is no longer fixed: it shifts in response to the measurement itself. This is Goodhart's Law, and in AI evaluation it is not a metaphor but a demonstrable, recurring phenomenon.

Concrete examples. The Foundation Model Transparency Index (FMTI) saw scores jump dramatically after its first release, not because companies became more transparent, but because they learned *which specific indicators* were measured and optimized their disclosures accordingly. Chatbot Arena has faced concerns about strategic submission: developers can selectively deploy models optimized for the types of queries that appear on the platform. And benchmark contamination — where training data includes evaluation items — is rational behavior for a developer whose market value depends on leaderboard position (Section 1.3.3).

Manheim and Garrabrant (2018) identify four variants of the Goodhart effect, each with a distinct causal mechanism:

1. **Regressional Goodhart**: The proxy $\hat{u}$ and the true objective u are correlated but not identical. Optimizing $\hat{u}$ overshoots because extreme values of $\hat{u}$ tend to arise from noise, not from genuinely extreme u. This is the measurement error story from Chapter 5: a benchmark with $\alpha = 0.7$ captures 70% true variance and 30% noise, so selecting the top-scoring model partially selects for lucky noise.
2. **Extremal Goodhart**: The relationship between $\hat{u}$ and u that holds in the bulk of the distribution breaks down in the tails. A benchmark that reliably ranks typical models may fail catastrophically when applied to a model specifically engineered to maximize the benchmark score.
3. **Causal Goodhart**: The proxy and the objective share a common cause. Optimizing the proxy can break this causal link. If "passes safety benchmark" and "is actually safe" are both caused by "was trained carefully," a developer can find shortcuts to the benchmark that bypass the common cause.
4. **Adversarial Goodhart**: An agent actively exploits the gap between proxy and objective. This is the regime that dominates in competitive AI evaluation, and the focus of this chapter.

The shift from the earlier measurement chapters to this one is a shift in the source of the threat. Measurement error (Chapter 5) is stochastic. Validity threats (Section 1.3) are systematic but unintentional. Strategic manipulation is *deliberate and adaptive*: the DGP itself changes in response to the measurement. This connects to the literature on performative prediction (Perdomo et al. 2020), where the act of deploying a model changes the distribution it operates on, and to strategic classification (Hardt et al. 2016), where agents manipulate their features to achieve favorable outcomes.

Underlying all four variants is a single structural fact: the *construct* we care about — say, "mathematical capability" — is always broader than any *distribution* of tasks we can actually pose. We instantiate "math ability" with one particular distribution over problems, and a rational builder optimizes for that distribution, not for the construct behind it. The rest of this chapter takes this gap as its organizing question. Rather than first asking *which mechanism* an evaluator should use, we ask *what object* the evaluator must keep private to preserve the meaning of the score (Section 11.2). That question orders the chapter: we escalate from hiding the realized test set (Section 11.4), to hiding the distribution that generates it (Section 11.5), to publishing the construct itself alongside public proxies (Section 11.6).

11.2 What Are We Trying to Hide?

Before choosing a mechanism, an evaluator should answer a prior question: *what object are we trying to keep private?* Different protocols protect different things, and once we name the object, the choice between deterministic and randomized machinery often settles itself. We distinguish three objects, nested from concrete to abstract.

Definition: Construct, Distribution, Set

- A **construct** $\mathcal{C}$ is the capability we intend to measure — e.g., "mathematical reasoning." It is a latent attribute (Chapter 1), not a finite object.
- A **distribution** D is an *operationalization* of $\mathcal{C}$: a way of generating tasks that we accept as measuring it. FrontierMath is a sample from one such distribution; the process that could generate *another* FrontierMath-like exam is D itself.
- A **set** $S \sim D$ is a realized, finite sample of tasks — the specific items on today's exam.

A single construct admits *many* distributions. GSM8K and FrontierMath both claim to measure mathematical capability, yet they induce very different task distributions D' and D; both are operationalizations of the same $\mathcal{C}$. This multiplicity is not a defect — it is exactly what lets us reason about validity and, eventually, what lets us publish a public proxy while keeping a private benchmark (Section 11.6).

The source of the incentive problem. The construct is always broader than any distribution that operationalizes it: $\mathcal{C}$ is an idealization, D a finite and fallible instrument. We *say* we measure "math ability" ($\mathcal{C}$) but we *contract on* performance under one distribution (D). A strategic builder optimizes for D, not $\mathcal{C}$. This $\mathcal{C} \supsetneq D$ gap is the engine behind every Goodhart variant of Section 11.1.

Connecting to the formal model. The game-theoretic notation of this chapter is a formalization of these three objects. The **social-relevance distribution** μ that weights the evaluator's utility $u_E(\theta) = \mathbb{E}_{f\sim\mu}[f(\theta)]$ — how much each task actually matters in deployment — is the construct $\mathcal{C}$ fully operationalized, "what we actually care about" (the uniform weighting $\mu = \text{Uniform}(F)$ is the simplest special case). The sampling distribution π_E is the operationalizing distribution D, and a realized draw $S = M(F_E, \omega)$ is the set S. In this language the incentive problem is simply the statement $\pi_E \neq \mu$: the distribution we can sample is a biased, narrow shadow of the construct we care about.

With the objects named, mechanisms sort cleanly by *what they hide*.

Table 11.1
Mechanisms classified by the object they keep private

Mechanism	Object hidden	What leaks	Durable?	Deterministic OK?
Static public benchmark	nothing	S, and hence D	no	—
Reusable holdout / Ladder (Section 11.4.2)	the set S	score signal, slowly	for a while	yes
Dynamic benchmark (Section 11.4.3)	today's S	D, over rounds	no	yes

Mechanism	Object hidden	What leaks	Durable?	Deterministic OK?
Randomized evaluation (Section 11.5)	the distribution D	D, slowly	yes, with correction	no
Construct-validity protocol (Section 11.6)	D and S	only proxy D'	yes	either

Two columns deserve comment. *Deterministic OK?* records the chapter's central reconciliation: deterministic machinery is fine when the object you protect is the *set* (a fixed holdout can be kept secret and reused), but it fails when the object is the *distribution* — Section 11.5 shows that hiding D *requires* randomization. The right mechanism depends on the object, not on a blanket preference for randomness. *Durable?* records whether the protection survives a builder who plays the game many times.

An orthogonal axis: how much leaks. Independent of *which* object you hide is *how much* you reveal about it per round. The Ladder algorithm (Section 11.4.2) is best understood as a control on this second axis: it can wrap either a hidden set or a hidden distribution, releasing a sliver of signal only when a submission genuinely improves. "What to hide" and "how much to leak" are separate design decisions, and conflating them is what makes the deterministic-versus-randomized debate feel irreconcilable.

Three properties. Naming the objects lets us state what a good evaluator should guarantee. We return to these in Section 11.9, but they motivate the whole chapter:

1. **Validity.** The reported score carries the intended construct interpretation — it tracks $\mathcal{C}$, not merely D (Chapter 1).
2. **Durability.** The benchmark stays informative over time, because D (and ideally S) remains private and therefore hard to game.
3. **Faithful execution.** The evaluator actually runs the promised procedure — the same sampling, scoring, and reporting for everyone, with no special treatment. This merges what one might separately call *neutrality* and *integrity*: both reduce to honestly executing the announced protocol.

Roadmap. The chapter now escalates through the objects. Section 11.4 hides the *set* S — reusable holdouts, the Ladder algorithm, and dynamic benchmarks — and shows where each stops working. Section 11.5 hides the *distribution* D through randomization, develops the information-variance tradeoff that hiding D creates, and resolves it with distribution correction. Section 11.6 then turns the $\mathcal{C} \supsetneq D$ gap from a liability into a tool: by publishing the construct and public proxies while keeping D and S private, an evaluator can certify capability without ever exposing the benchmark.

11.3 The Evaluation Game

We now formalize the strategic interaction between an evaluation designer and a model builder as a Stackelberg game. The designer moves first by committing to an evaluation mechanism; the builder responds by training a model. The key question: under what conditions can a benchmark mechanism incentivize builders to improve performance on the *full* task distribution, rather than merely optimizing for the specific tasks being measured?

11.3.1 Setup: Evaluator and Builder

Let Θ denote the space of all possible models. For a given model $\theta \in \Theta$, let $f(\theta) \in [0, 1]$ denote the model's performance on task f. The universe of all possible tasks is the finite set F with $|F| = N$.

Definition: Evaluation Designer's Utility

The evaluation designer's utility for a model θ is its performance averaged under the **social-relevance distribution** μ over tasks:

$$u_E(\theta) = \sum_{f \in F} f(\theta)\, \mu(f) = \mathbb{E}_{f \sim \mu}[f(\theta)]$$

where $\mu(f)$ is how socially relevant task f is — e.g., the probability that f is encountered in real deployment. The designer acts as a social planner seeking models that perform broadly across what matters, rather than narrowly. The uniform weighting $\mu = \text{Uniform}(F)$, under which $u_E(\theta) \propto \sum_f f(\theta)$, is the simplest special case.

Each party acquires tasks by sampling from F: the designer draws tasks according to π_E over F, forming the evaluation set F_E, while the builder draws according to π_M, forming the training set F_M. In practice, π_M is approximately uniform (builders have broad access to training data), while π_E is typically biased toward tasks that are easy to construct, grade, or that reflect particular evaluation priorities — in general $\pi_E \neq \mu$.

Definition: Stackelberg Benchmark Game

The game proceeds in three stages, parameterized by a mechanism (M, r) where M is a sampling function and $r : \Theta \times \mathcal{P}(F) \to \mathbb{R}$ is a reward function:

1. **Designer's move (ex-ante)**: The evaluation designer publishes the mechanism (M, r). If M is randomized, the designer privately draws randomness ω and computes $S^* = M(F_E, \omega)$, withholding the realization. If M is deterministic, $S^* \subseteq F_E$ is fixed.
2. **Builder's move**: The builder observes the mechanism (M, r) but not the realized set S^*, and selects a model $\theta^* \in \Theta$.
3. **Evaluation stage (ex-post)**: The designer publishes the score $r(\theta^*, S^*)$. The builder's payoff is $r(\theta^*, S^*)$; the designer's payoff is $u_E(\theta^*)$.

The misalignment between the builder's incentive (performance on S^*) and the designer's objective (performance on all of F) creates the Goodhart problem.

11.4 Hiding the Set

The mildest thing an evaluator can keep private is the *realized test set* S. The construct and even the sampling distribution may be public; only the specific items on today's exam are secret. This is the regime of *held-out* test sets, and — crucially — it is a regime where *deterministic* machinery works: a fixed, secret holdout, reused carefully across many submissions, can preserve a leaderboard for a long time. The catch is that every score computed on S leaks a little information about it, so the central design problem is *bounding that leak*. We first see why the naive version (publishing S) fails, then how the Ladder algorithm hides S while bounding leakage, and finally why refreshing S — dynamic benchmarking — does not reach the harder goal of hiding D.

11.4.1 The Static Benchmark and Why It Fails

Consider first a *transparent* mechanism: the builder observes the realized evaluation set $S^* \subseteq F_E$ before choosing a model. A published, fixed benchmark is the canonical example.

Proposition: Failure of Transparent Mechanisms

If the builder observes the realized set S^* and the reward is non-degenerate on it (e.g., additive $r(\theta, S) = \sum_{f \in S} f(\theta)$), the builder's best response is:

$$\theta^* \in \arg\max_{\theta \in \Theta} \sum_{f \in S^*} f(\theta)$$

Whenever $|S^*| < |F|$, there exist a task universe F and social-relevance weighting μ for which this is socially suboptimal: $u_E(\theta^*) < \max_\theta u_E(\theta)$. The mechanism provides *no incentive* for performance on tasks outside S^*.

Proof. The builder observes S^* and maximizes the known objective, so tasks in $F \setminus S^*$ do not affect the score. Place a socially decisive task $f' \notin S^*$ (some $\mu(f') > 0$ with f' maximized away from θ^*); then θ^* misses it. □

This is exactly Goodhart's adversarial variant: the builder exploits the *observed* structure of the evaluation to specialize, the construct underrepresentation problem of Section 1.3.2 seen through a strategic lens.

It is worth being precise about *what* fails here. The culprit is **transparency**, not determinism. A benchmark whose sampling rule is deterministic but whose realized set S^* is kept *hidden* does not fall under the proposition: the builder is then Bayesian over S^*, and the outcome is governed by the single-sample alignment bound of Section 11.5.1 via the builder's belief $\tilde{\pi}_E$. Randomization is one way to enforce the necessary secrecy; a hidden but deterministic holdout is another — with the crucial caveat that *any leak collapses it back to the transparent*

case (the failure mode the Ladder of Section 11.4.2 is designed to slow). We visualize the transparent-versus-randomized contrast in Section 11.5.1.

11.4.2 The Ladder Algorithm and Reusable Holdouts

If a static *public* set fails because S is exposed, the obvious fix is to keep S secret and let builders query it only through scores. But scores themselves leak: a builder who submits many models against a fixed holdout can overfit it through the published numbers alone, even without ever seeing an item — the *adaptive data analysis* problem. Blum and Hardt (2015)'s **Ladder** algorithm bounds this leak. It releases a new score only when a submission improves on the best score so far by more than a noisy margin; otherwise it re-reports the previous best. Because most submissions are not genuine improvements, almost no information about S flows back to the builder.

Algorithm: The Ladder

Maintain a running best published score $b_0 = -\infty$. For each submission θ_t ($t = 1, 2, \ldots$):

1. Compute the holdout score $r_H(\theta_t) = \frac{1}{|S|}\sum_{f \in S} f(\theta_t)$ on the secret set S.
2. Draw noise $\xi_t \sim \text{Laplace}(0, 1/\varepsilon)$ and a threshold $\eta > 0$.
3. **If** $r_H(\theta_t) > b_{t-1} + \eta + \xi_t$: publish $b_t \leftarrow \lfloor r_H(\theta_t) \rceil_\eta$ (round to step η). **Else**: publish $b_t \leftarrow b_{t-1}$ (no new information).

We can extend this into a mechanism that gates information on the builder's *behavior* rather than only on improvement, by comparing the secret holdout against a public reference set.

Definition: Noise-Gated Holdout

The evaluator publishes a reference set $S_0 \subset F$ of size n_0, sampled from π_E. At each round t:

1. Draw a holdout set S_t of size k from π_E and noise $\xi_t \sim \text{Laplace}(0, 1/\varepsilon)$.
2. Compute the reference score $r_T(\theta_t) = \frac{1}{n_0}\sum_{f \in S_0} f(\theta_t)$ and the holdout score $r_H(\theta_t) = \frac{1}{k}\sum_{f \in S_t} f(\theta_t)$.
3. **Threshold test**: If $|r_H(\theta_t) + \xi_t - r_T(\theta_t)| < \eta$, publish $r_T(\theta_t)$ ("silent" round). Otherwise, publish $r_H(\theta_t) + \xi_t$ ("alarm" round).

The mechanism is **self-correcting**: it reveals less information precisely when the builder is behaving well ($\Delta_{\text{game}} \approx 0$, so the alarm fires only due to noise, with probability $p_{\text{alarm}} = e^{-\varepsilon\eta}$), and reveals more when the builder is gaming (the score discrepancy reliably triggers the alarm). The effective leakage per round for a non-gaming builder is:

$$\lambda_{\text{out}}^{\text{holdout}} \approx e^{-\varepsilon\eta} \cdot k$$

Setting $\varepsilon\eta = 3$ yields a $\sim 20\times$ slowdown in the builder's learning rate.

This is a differential-privacy guarantee in disguise (Dwork et al. 2015; Dwork and Roth 2014): the Laplace threshold is the Sparse Vector mechanism, and it makes the published transcript

insensitive to any single held-out item, bounding the max-information between the builder's observations and S. Hiding the set is, in this sense, a solved problem: a secret S can be reused for a bounded number of informative queries.

Why Hiding the Set Is Not Enough

A perfectly hidden, leak-bounded set protects only S. If the *distribution* $D = \pi_E$ is biased and fixed, a builder who never sees a single item can still specialize to it — because D is public, or quickly inferred. The builder's optimal model given their belief $\hat{\pi}_E$ about the distribution is $\theta^* = \arg\max_\theta \mathbb{E}_{f\sim\hat{\pi}_E}[f(\theta)]$, regardless of how little leaks about any particular S. Hiding the set caps overfitting to particular *items*; it does nothing about overfitting to the *distribution*. That requires hiding D itself — the subject of Section 11.5.

11.4.3 Dynamic Benchmarks

A popular alternative to a single hidden holdout is to *refresh* it: draw a fresh set $S' \sim D$ each period and evaluate on it immediately. DynaBench (Kiela et al. 2021), LiveBench (White et al. 2025), and the rolling battles of Chatbot Arena (Chiang et al. 2024) are all, in this framing, *repeated sampling of $S' \sim D$ with immediate disclosure.* Refreshing defeats item-level contamination: yesterday's leaked items are worthless because tomorrow's set is new.

But dynamic refresh hides the *set*, not the *distribution*, and two limitations follow.

First, *it does not hide D.* Each disclosed S' is a fresh sample from the same D; across rounds the builder accumulates samples and can estimate D to arbitrary precision (the posterior-concentration argument of Section 11.5.3.2). A reader may ask: why repeatedly draw a new S' at all — why not build two sets S and S' once, publish S', and permanently hold out S? That design is exactly the reusable holdout of Section 11.4.2, and the held-out S is consumed at the Ladder-bounded leak rate; refreshing merely trades a slowly-leaking fixed set for a stream of fully-disclosed ones. Either way D becomes public or quickly learned. Durability against *distributional* gaming requires hiding D, not cycling S.

Second, *if D deviates from the construct, refreshing cannot help.* Suppose D over-weights, say, competition-style problems relative to the construct weighting μ. Every $S' \sim D$ inherits that bias. A builder who tracks D — easy, since it leaks — specializes to it, and the resulting misalignment does not vanish with the refresh rate: it is governed by how far D sits from μ, on the order of $\mathrm{TV}(\mu, D)$, no matter how often S is renewed (Exercise 9). Dynamic benchmarks fix contamination; they do not fix construct misalignment.

These two failures — a learnable D and a biased D — are exactly what the next sections address: Section 11.5 hides D and corrects it toward μ, and Section 11.6 confronts the $D \neq \mathcal{C}$ gap directly.

11.5 Hiding the Distribution

Hiding the set caps how much a builder can overfit *particular items*, but it leaves the *distribution* $D = \pi_E$ exposed — and a public distribution is a public target. To incentivize broad capability we must keep D itself uncertain to the builder. This section shows that hiding D (i) *requires* randomization — no deterministic mechanism suffices — (ii) creates an unavoidable tradeoff between score variance and information leakage, and (iii) is made durable by *correcting* D toward the construct μ faster than the builder can learn it.

11.5.1 One-Shot Alignment via Randomization

While transparent mechanisms fail, randomization offers a path forward. When the builder faces genuine uncertainty about which task will be scored, the strategic landscape changes fundamentally. The simplest such mechanism scores a single random task.

DEFINITION: SINGLE-SAMPLE MECHANISM

$S(F_E, \omega)$ draws one task s uniformly from F_E, and the reward is $r(\theta, \{s\}) = s(\theta)$. The builder, not knowing which task will be drawn, is Bayesian: it holds a belief $\tilde{\pi}_E$ over which tasks the evaluator might score, where $\tilde{\pi}_E(f)$ is the builder's subjective probability that f is the sampled task.

Because the builder cannot identify the realized task, it can only optimize against its *belief* $\tilde{\pi}_E$.

PROPOSITION: BEST RESPONSE UNDER A SINGLE-SAMPLE MECHANISM

Under the single-sample mechanism, the Bayesian builder's best response is

$$\theta^M \in \arg\max_{\theta \in \Theta} \mathbb{E}_{f \sim \tilde{\pi}_E}[f(\theta)].$$

The builder optimizes broadly, over its whole belief about what might be tested, rather than for any fixed set.

The value of this is captured by a single bound: the evaluator's loss is controlled by how far the builder's belief sits from social relevance.

THEOREM: SINGLE-SAMPLE ALIGNMENT

Under the single-sample mechanism, the evaluator's suboptimality is bounded by the total variation distance between the builder's belief and the social-relevance distribution:

$$\max_{\theta \in \Theta} u_E(\theta) - u_E(\theta^M) \;\le\; 4B \cdot \mathrm{TV}(\mu, \tilde{\pi}_E),$$

where $B = \sup_{f \in F;\, a,b \in \Theta} |f(a) - f(b)|$ bounds the spread of task scores.

Proof sketch. The builder maximizes $\mathbb{E}_{\tilde{\pi}_E}[f(\theta)]$ while the evaluator cares about $\mathbb{E}_{\mu}[f(\theta)] = u_E(\theta)$. Adding and subtracting the belief-weighted objective at the builder's and the opti-

mal model and using optimality of θ^M for the former bounds the gap by $2\sup_\theta |\mathbb{E}_\mu[f(\theta)] - \mathbb{E}_{\tilde{\pi}_E}[f(\theta)]|$; since scores lie in an interval of width $\leq B$, this discrepancy is at most $2B\,\mathrm{TV}(\mu, \tilde{\pi}_E)$. □

This bound is the chapter's central one-shot result, and it makes the privacy intuition exact: *misalignment is bounded by the distance between what the builder believes will be tested and what society actually cares about.* Two corollaries pin down the endpoints.

Corollary: Perfect Alignment

If the builder's belief matches social relevance, $\tilde{\pi}_E = \mu$, then $\mathrm{TV} = 0$ and θ^M maximizes u_E exactly. A well-informed builder who believes the evaluator is equally well-informed is perfectly aligned. (The uniform special case $\mu = \mathrm{Uniform}(F)$ recovers the familiar statement that the builder's expected reward is proportional to u_E.)

Corollary: Randomization Strictly Broadens Incentives

If $F_E \subset F_M$ and $\tilde{\pi}_E(f) > 0$ for some $f \notin F_E$, the single-sample mechanism incentivizes the builder over a strictly larger set of tasks than *any* transparent mechanism — including tasks the evaluator cannot even sample, as long as the builder believes they might be tested.

The practical upshot: under randomization the builder's best move is to optimize for its honest belief about what matters, which — for a well-resourced builder whose belief is broad — means training broadly rather than gaming a known set. A boundedly rational builder that can only estimate $\mathbb{E}_{\tilde{\pi}_E}[f(\theta)]$ from a finite sample $F_M \sim \pi_M$ does importance-weighted **empirical risk minimization**; by uniform convergence (Vapnik 1998) its worst-case guarantee is monotone in sample size, so it has no statistical reason to discard tasks — no incentive to "game" by narrowing.

11.5.2 Discussion: What Randomization Buys

The one-shot alignment result reveals a fundamental tension with the statistical efficiency goals of earlier chapters. In Section 6.2.1 and Section 6.2.2, we showed that *targeted* evaluation — choosing items to maximize Fisher information about θ — is statistically optimal. But targeted evaluation requires revealing information about which tasks the evaluator considers informative, which creates exploitable structure.

Randomized evaluation is not optimal for *estimation precision* but is optimal for *incentive alignment*. This tension between statistical efficiency and strategic robustness is a recurring theme:

- Computerized adaptive testing (Section 6.2.2) reveals the evaluator's information about ability through the item selection strategy, making the evaluator's priorities transparent.
- Bayesian persuasion (Kamenica and Gentzkow 2011) provides the information design framework: the evaluator is a sender choosing an information structure, and the builder

is a receiver who updates and best-responds. The evaluator's problem is to design a signal that induces the builder to take the action the evaluator prefers.

The resolution, as we develop in the next two sections, is that the evaluator must *invest in correction* to make the inevitable information leakage harmless.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: deterministic-vs-randomized
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# --- Setup ---
N = 50           # total tasks in universe F
n_eval = 10      # tasks in evaluation set
n_rounds = 200   # simulation rounds

# Social-relevance distribution mu over tasks (general, non-uniform)
mu = np.random.uniform(0.5, 1.5, N)
mu /= mu.sum()

# --- Transparent mechanism: builder observes the realized set S* ---
# Builder puts all effort into the known evaluation tasks
det_eval_set = np.random.choice(N, n_eval, replace=False)
det_perf = np.zeros(N)
det_perf[det_eval_set] = 0.95  # high on evaluated tasks
det_perf[~np.isin(np.arange(N), det_eval_set)] = 0.2  # low elsewhere

```

```
# --- Randomized mechanism: builder faces uncertainty ---
# Under randomized eval, builder's best response is to train broadly
rand_perf = np.full(N, 0.65)  # uniform effort across all tasks

# --- Evaluate ---
# True utility u_E = E_{f ~ mu}[performance] (social-relevance weighted)
det_utility = float(det_perf @ mu)
rand_utility = float(rand_perf @ mu)

# Eval score = average on sampled tasks
n_trials = 500
det_scores = np.array([det_perf[np.random.choice(N, n_eval, replace=False)].mean()
                       for _ in range(n_trials)])
rand_scores = np.array([rand_perf[np.random.choice(N, n_eval, replace=False)].mean()
                        for _ in range(n_trials)])

# --- Plot ---
fig, axes = plt.subplots(1, 3, figsize=(6, 2))
colors = ['#5B8DEE', '#E8637A', '#45BF7C', '#F0A35C', '#B07CD8']

# Panel 1: Task-level performance
ax = axes[0]
sorted_idx_det = np.argsort(-det_perf)
ax.bar(range(N), det_perf[sorted_idx_det], color=colors[1], alpha=0.7, width=1.0)
ax.set_xlabel('Task (sorted)')
ax.set_ylabel('Performance')
ax.set_title('Transparent')
ax.set_ylim(0, 1.05)
ax.axhline(y=det_perf.mean(), color='k', ls='--', lw=0.8)

ax = axes[1]
ax.bar(range(N), rand_perf[np.argsort(-rand_perf)], color=colors[0], alpha=0.7,
 ↪  width=1.0)
ax.set_xlabel('Task (sorted)')
ax.set_title('Randomized')
ax.set_ylim(0, 1.05)
ax.axhline(y=rand_perf.mean(), color='k', ls='--', lw=0.8)

# Panel 3: Utility comparison
ax = axes[2]
bar_x = [0, 1]
bar_vals = [det_utility, rand_utility]
bar_colors = [colors[1], colors[0]]
bars = ax.bar(bar_x, bar_vals, color=bar_colors, width=0.6, alpha=0.8)
ax.set_xticks(bar_x)
ax.set_xticklabels(['Transp.', 'Rand.'])
ax.set_ylabel('True utility $u_E$ ( -weighted)')
ax.set_title('Alignment')
ax.set_ylim(0, 0.85)
```

```
for b, v in zip(bars, bar_vals):
    ax.text(b.get_x() + b.get_width()/2, v + 0.02, f'{v:.2f}', ha='center',
    ↪ va='bottom')

plt.tight_layout()
plt.show()
```

Figure. Under a transparent mechanism (left), the builder observes the evaluation set and concentrates effort on the $k = 10$ known tasks, neglecting the rest — high benchmark score, low true utility. Under a randomized mechanism (center), facing uncertainty about what will be scored, the builder spreads effort broadly. The right panel compares true utility $u_E = \mathbb{E}_{f\sim\mu}[\text{performance}]$ under the social-relevance weighting μ: randomization yields substantially higher utility despite lower peak performance on any single task.

11.5.3 The Information-Variance Tradeoff

The single-sample alignment bound (Section 11.5.1) showed that randomization aligns incentives in a *one-shot* game, up to the builder's belief error $\mathrm{TV}(\mu, \tilde{\pi}_E)$. But benchmarks operate as *repeated games*: weekly leaderboards, monthly submissions, quarterly benchmark releases. In repeated evaluation, randomization alone fails because information *leaks* — the builder's belief $\tilde{\pi}_E$ drifts toward the true π_E, and if $\pi_E \neq \mu$ the bound degrades.

11.5.3.1 Repeated Evaluation and Information Leakage

DEFINITION: REPEATED EVALUATION GAME

At each round $t = 1, 2, \ldots$:

1. The evaluator draws a fresh evaluation set $F_E^{(t)}$ by sampling from F according to π_E.
2. The evaluator samples k tasks from $F_E^{(t)}$ via mechanism M and publishes the score.
3. The builder observes the k evaluated tasks, updates their estimate $\hat{\pi}_{E,t}$ of the evaluation distribution, and selects model θ_t.

For a size-k subset drawn without replacement from F_E with per-task score variance σ^2, the average reward has the finite-population variance

$$\mathrm{Var}(\hat{r}_k) = \frac{\sigma^2}{k} \cdot \frac{|F_E| - k}{|F_E| - 1} \approx \frac{\sigma^2}{k} \quad (k \ll |F_E|).$$

To reliably distinguish two models with performance gap Δ at 95% confidence requires $k \geq 4\sigma^2/\Delta^2$. As models improve and converge ($\Delta \to 0$), the required sample size grows quadratically. This creates inexorable market pressure to increase k.

But increasing k to reduce variance simultaneously increases the rate at which the builder accumulates information about π_E. The builder's *information set* at time t is $\mathcal{I}_t = \bigcup_{i=1}^{t-1} S_i$, where S_i is the set of tasks sampled in round i. The *leakage* is $L_t = |\mathcal{I}_t|/|F|$.

11.5.3.2 Posterior Concentration and Incentive Misalignment

As the builder observes sampled tasks across rounds, their estimate $\hat{\pi}_{E,t}$ of the evaluation distribution concentrates around the true π_E. By standard results in Bayesian nonparametrics, the posterior concentration rate is $O(d_{\text{eff}} \log m/m)$ where m is the number of observations and d_{eff} is the effective dimension of the distribution class.

Proposition: Incentive Misalignment Under Distribution Learning

Let $\hat{\pi}_{E,t}$ denote the builder's posterior mean estimate of π_E at time t. The builder's optimal strategy is $\theta_t^* = \arg\max_\theta \mathbb{E}_{f\sim\hat{\pi}_{E,t}}[f(\theta)]$. Then:

1. **Initial alignment**: When the prior over π_E is diffuse, the builder optimizes broadly, approximating performance on the full universe F.
2. **Posterior concentration leads to specialization**: As $\hat{\pi}_{E,t}$ concentrates around π_E:

$$\lim_{t\to\infty} \theta_t^* = \arg\max_{\theta\in\Theta} \mathbb{E}_{f\sim\pi_E}[f(\theta)]$$

   If π_E is non-uniform, the builder specializes to high-density regions at the expense of broad capability.

This is the core tension: *privatization works in one-shot settings, but repeated evaluation reveals information that enables strategic specialization.* The builder cannot identify a fixed evaluation set (since $F_E^{(t)}$ is resampled each round), but from repeated observations the builder learns π_E itself — the *distribution* from which evaluation tasks are drawn.

11.5.3.3 The Pareto Frontier

Define the **residual misalignment** at time t as the evaluator's regret from the builder optimizing for its (drifting) belief rather than for u_E directly:

$$\Delta_t := \max_{\theta\in\Theta} u_E(\theta) - u_E(\theta_t).$$

This measures the gap between the best model society could get and the model the builder actually fields once it has begun to learn π_E. When the builder's effective belief equals μ — in particular when $\pi_E = \mu$ so that what leaks is exactly the construct — $\Delta_t = 0$.

No evaluation strategy can simultaneously achieve low variance and low leakage. Variance decreases in k; leakage increases in k. Varying k traces a Pareto frontier where the evaluator must choose their preferred operating point.

```
#| label: pareto-frontier
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# --- Simulation parameters ---
N = 200          # task universe size
n_rounds = 200   # number of rounds
k_values = [5, 10, 20, 50, 100]
n_trials = 20    # Monte Carlo trials

# Social-relevance distribution mu (the construct, generally non-uniform)
rng0 = np.random.default_rng(0)
mu = rng0.uniform(0.5, 1.5, N)
mu /= mu.sum()

# Biased evaluator sampling distribution pi_E: overweight first 40% of tasks
pi_E_true = np.ones(N) / N
pi_E_true[:int(0.4 * N)] *= 3.0
pi_E_true /= pi_E_true.sum()

colors = ['#5B8DEE', '#45BF7C', '#F0A35C', '#E8637A', '#B07CD8']

fig, axes = plt.subplots(1, 3, figsize=(6, 2))

for ki, k in enumerate(k_values):
    all_variance = np.zeros((n_trials, n_rounds))
    all_kl = np.zeros((n_trials, n_rounds))
    all_misalign = np.zeros((n_trials, n_rounds))

    for trial in range(n_trials):
        # Builder starts with uniform prior (Dirichlet(1,...,1))
        counts = np.ones(N)

        for t in range(n_rounds):
            # Evaluator samples k tasks from pi_E
            tasks = np.random.choice(N, size=k, p=pi_E_true, replace=True)

            # Score variance: use task values as performance
            task_scores = np.random.normal(0.5, 0.2, k)
            all_variance[trial, t] = np.var(task_scores)

            # Builder updates posterior
            for task in tasks:
                counts[task] += 1

            # Builder's estimate of pi_E
```

```
            pi_hat = counts / counts.sum()

            # KL divergence from true pi_E
            mask = pi_E_true > 0
            kl = np.sum(pi_E_true[mask] * np.log(pi_E_true[mask] / pi_hat[mask]))
            all_kl[trial, t] = kl

            # Misalignment relative to the construct mu (not uniform).
            # Builder can only exploit the bias it has actually learned:
            tv = 0.5 * np.sum(np.abs(pi_hat - mu))           # belief vs construct
            bias_tv = 0.5 * np.sum(np.abs(pi_E_true - mu))   # how far pi_E sits from
    ↪ mu
            all_misalign[trial, t] = min(tv, bias_tv)

    col = colors[ki % len(colors)]
    rounds = np.arange(n_rounds)

    # Panel 1: Evaluation variance (averaged over rounds, show as function of t)
    avg_var = all_variance.mean(axis=0)
    axes[0].plot(rounds, np.convolve(avg_var, np.ones(10)/10, mode='same'),
                 color=col, label=f'k={k}', lw=1.2)

    # Panel 2: KL divergence (builder's learning of pi_E)
    avg_kl = all_kl.mean(axis=0)
    axes[1].plot(rounds, avg_kl, color=col, label=f'k={k}', lw=1.2)

    # Panel 3: Misalignment
    avg_mis = all_misalign.mean(axis=0)
    axes[2].plot(rounds, avg_mis, color=col, label=f'k={k}', lw=1.2)

axes[0].set_xlabel('Round')
axes[0].set_ylabel('Score variance')
axes[0].set_title('Evaluation noise')
axes[0].legend(ncol=2, loc='upper right')

axes[1].set_xlabel('Round')
axes[1].set_ylabel('KL divergence')
axes[1].set_title("Builder's learning")

axes[2].set_xlabel('Round')
axes[2].set_ylabel('Misalignment')
axes[2].set_title('Incentive drift')

plt.tight_layout()
plt.show()
```

Figure. The information-variance Pareto frontier across $k \in \{5, 10, 20, 50, 100\}$ over 200 rounds with $|F| = 200$ tasks. Left: evaluation noise (score variance) is lower for larger k. Center: the builder learns π_E faster with larger k, as measured by KL divergence between the builder's

posterior and the true π_E. Right: as the builder's posterior concentrates, incentive misalignment grows. No single k achieves both low noise and low misalignment — a fundamental tradeoff.

11.5.4 Restoring Alignment

The Pareto frontier of the previous section seems like an impasse: any choice of k sacrifices either measurement precision or incentive alignment. The resolution comes from recognizing that the *evaluator is also a learner*. If the evaluator corrects their distribution over time, the information the builder accumulates becomes obsolete.

11.5.4.1 Distribution Correction

The right target for correction is the construct μ, not uniformity. The evaluator does not reweight blindly toward a flat distribution; it discovers *which regions it under-covers* and fills them.

ASSUMPTION: GAP-TARGETED GAUSSIAN CORRECTION

Embed tasks in a feature space, so each $f \in F$ has a location z_f. At each round t, the evaluator:

1. **Discovers a gap**: samples a location ν_t with probability proportional to the local deficit relative to the social-relevance distribution μ,

$$P(\nu_t = f \mid \pi_E^{(t-1)}) \;\propto\; [\mu(f) - \pi_E^{(t-1)}(f)]_+.$$

2. **Fills locally**: mixes in a Gaussian bump centered at the gap,

$$\pi_E^{(t)} = (1-\rho)\,\pi_E^{(t-1)} + \rho\, g_{\sigma_c}(\cdot\,;\nu_t), \qquad g_{\sigma_c}(f;\nu) \propto \exp(-\|z_f - z_\nu\|^2/2\sigma_c^2),$$

normalized over F. The parameters are the *correction rate* $\rho \in (0,1]$ and the *correction bandwidth* $\sigma_c > 0$. The process drives $\pi_E^{(t)} \to \mu$.

In practice, evaluators discover these gaps through external channels: *incident reports* from users who hit model failures the benchmark missed; *systematic audits* of coverage across capability dimensions (languages, modalities, reasoning types); *A/B testing* of rankings under different subsets; and *improving acquisition capacity* over time. Each channel fills an under-covered region rather than flattening the distribution — correction aims at μ, the construct, not at uniformity.

THEOREM: MISALIGNMENT UNDER GAP-TARGETED CORRECTION

Under gap-targeted Gaussian correction, with $m_t = k \cdot \min(t, 1/\rho)$ the builder's effective sample size, the

expected residual misalignment satisfies

$$\mathbb{E}[\Delta_t] \le \min\left(\underbrace{\frac{m_t}{m_t + |F|}}_{\text{estimation-limited}} \;,\; \underbrace{(1-\rho)^t \sqrt{0.5\, D_0} + \varepsilon(\rho, \sigma_c)}_{\text{correction-limited}} \right),$$

where $D_0 = \mathrm{KL}\big(\pi_E^{(0)} \,\|\, \mu\big)$ is the initial bias of the evaluator's distribution from the construct, and $\varepsilon(\rho, \sigma_c) = O\big(\sqrt{\rho} \cdot \mathrm{TV}(g_{\sigma_c}, \mu)\big)$ is fluctuation from the stochastic gap-discovery process (vanishing as $\sigma_c \to \infty$ or $\rho \to 0$).

The two terms capture complementary regimes — a *race between two learners*: the builder learns $\pi_E^{(t)}$ from observations while the evaluator corrects $\pi_E^{(t)}$ toward μ.

- **Estimation-limited** (t small): $m_t \approx kt$, so $\Delta_t \lesssim kt/|F|$. The builder has too few observations to identify the bias.
- **Correction-limited** (t large): $\Delta_t \lesssim (1-\rho)^t \sqrt{D_0/2} + \varepsilon$. The builder may know $\pi_E^{(t)}$ well, but correction has driven it toward μ, leaving almost nothing to exploit.

The crossover is at roughly $t^* \approx |F|/(k + |F|\rho)$: before t^* estimation limits the builder, after it correction does. Crucially, k appears only in the estimation term — a smaller k slows the builder's learning, buying the evaluator time to correct.

```
#| label: distribution-correction
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# --- Parameters ---
N = 200
n_rounds = 200
rho_values = [0, 0.01, 0.05, 0.1, 0.2]
n_trials = 20
sigma_c = 0.04                        # correction bandwidth

# Tasks embedded on [0, 1]
z = np.linspace(0, 1, N)

# Social-relevance distribution mu: smooth, non-uniform (two soft modes)
mu = (1.0
      + 0.9 * np.exp(-((z - 0.30) ** 2) / (2 * 0.08 ** 2))
      + 0.6 * np.exp(-((z - 0.75) ** 2) / (2 * 0.05 ** 2)))
mu /= mu.sum()

# Initial biased evaluator distribution: overweight the left third
pi_E_init = np.ones(N)
```

```
pi_E_init[:int(0.4 * N)] *= 3.0
pi_E_init /= pi_E_init.sum()

# Precompute Gaussian fill kernels: column nu is the bump centered at task nu
G = np.exp(-((z[:, None] - z[None, :]) ** 2) / (2 * sigma_c ** 2))
G /= G.sum(axis=0, keepdims=True)

colors = ['#888888', '#B07CD8', '#F0A35C', '#45BF7C', '#5B8DEE']

fig, axes = plt.subplots(1, 2, figsize=(6, 2))

for ri, rho in enumerate(rho_values):
    all_tv = np.zeros((n_trials, n_rounds))
    all_kl = np.zeros((n_trials, n_rounds))

    for trial in range(n_trials):
        pi_E = pi_E_init.copy()
        for t in range(n_rounds):
            if rho > 0:
                deficit = np.clip(mu - pi_E, 0, None)   # gap relative to construct mu
                if deficit.sum() > 1e-12:
                    nu = np.random.choice(N, p=deficit / deficit.sum())
                    pi_E = (1 - rho) * pi_E + rho * G[:, nu]

            all_tv[trial, t] = 0.5 * np.sum(np.abs(pi_E - mu))
            m = pi_E > 1e-15
            all_kl[trial, t] = np.sum(pi_E[m] * np.log(pi_E[m] / mu[m]))

    col = colors[ri]
    rounds = np.arange(n_rounds)
    label = f' ={rho}' if rho > 0 else ' =0 (no correction)'

    axes[0].plot(rounds, all_tv.mean(axis=0), color=col, label=label, lw=1.5)
    axes[1].plot(rounds, np.maximum(all_kl.mean(axis=0), 1e-10), color=col,
 ↪  label=label, lw=1.5)

axes[0].set_xlabel('Round')
axes[0].set_ylabel('Misalignment  TV( _E,  )')
axes[0].set_title('Residual misalignment')
axes[0].legend(fontsize=6)

axes[1].set_xlabel('Round')
axes[1].set_ylabel('KL( _E ‖  )')
axes[1].set_title('Distribution bias')
axes[1].set_yscale('log')
axes[1].legend(fontsize=6)

plt.tight_layout()
plt.show()
```

Figure. Alignment recovery under gap-targeted Gaussian correction toward a non-uniform construct μ, with $|F| = 200$. Left: residual misalignment $\mathrm{TV}(\pi_E^{(t)}, \mu)$ falls toward zero for all $\rho > 0$, faster for larger ρ; the $\rho = 0$ baseline (gray) stays biased. Right: the bias $\mathrm{KL}(\pi_E^{(t)} \,\|\, \mu)$ decays on a log scale, settling at a small floor set by the correction bandwidth σ_c (the $\varepsilon(\rho, \sigma_c)$ term of the theorem).

11.5.4.2 Optimal Evaluation Size

With distribution correction, the Pareto frontier from Section 11.5.3.3 collapses: each revealed task gives the builder only *transient* advantage, since the bias it reveals is corrected within $O(1/\rho)$ rounds. In steady state the builder's effective sample size saturates at $m_\infty = k/\rho$ (new observations arrive at rate k, old ones age out at rate ρ), so the estimation-limited term of the theorem above becomes a steady-state exploitation cost $k/(\rho|F| + k)$. Combining it with the evaluation variance σ^2/k gives the *asymptotic per-round loss*

$$\mathcal{L}(k) = \underbrace{\frac{\sigma^2}{k}}_{\text{evaluation variance}} + \underbrace{\frac{k}{\rho|F| + k}}_{\text{steady-state exploitation}}.$$

The first term decreases in k (more tasks, less noise); the second increases in k (more leakage) and decreases in ρ (faster correction shortens the exploitation window).

LEMMA: OPTIMAL SAMPLE SIZE

If $\rho|F| > \sigma^2$, the loss $\mathcal{L}(k)$ is minimized at

$$k^* = \frac{\sigma\,\rho|F|}{\sqrt{\rho|F|} - \sigma}.$$

If $\rho|F| \le \sigma^2$, $\mathcal{L}$ is strictly decreasing in k and the optimum is the largest admissible k (variance reduction dominates uniformly).

In the high-correction regime $\rho|F| \gg \sigma^2$, $k^* \approx \sigma\sqrt{\rho|F|}$, which gives clean comparative statics:

- k^* **grows with** $\sqrt{\rho}$: each revealed bias goes stale within $\sim 1/\rho$ rounds, so faster correction permits a larger evaluation set.
- k^* **grows with** σ: higher score noise demands more samples for a reliable signal.
- k^* **grows with** $\sqrt{|F|}$: a richer task universe dilutes any single leak.
- **The threshold** $\rho|F| > \sigma^2$ **is itself informative**: when correction reaches fewer new tasks per round than the noise scale, no interior optimum exists and the evaluator should simply maximize k. The capacity to identify and correct biases, ρ, is thus the key design lever — it both lifts this threshold and raises k^*. Investing in correction (coverage audits,

benchmark changelogs, incident-driven task additions) is the most effective move in benchmark design.

```
#| label: optimal-k-surface
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

sigma = 1.0          # per-task score noise scale
F_size = 2000        # |F|, the task universe
rho_values = [0.01, 0.02, 0.05, 0.1]

k_range = np.linspace(1, 40, 400)
colors = ['#E8637A', '#F0A35C', '#45BF7C', '#5B8DEE']

fig, ax = plt.subplots(1, 1, figsize=(3.5, 3))

for i, rho in enumerate(rho_values):
    a = rho * F_size                          # rho|F|
    loss = sigma**2 / k_range + k_range / (a + k_range)
    ax.plot(k_range, loss, color=colors[i], lw=1.5, label=f' ={rho}')

    if a > sigma**2:                          # interior optimum exists
        k_star = sigma * a / (np.sqrt(a) - sigma)
        loss_star = sigma**2 / k_star + k_star / (a + k_star)
        ax.plot(k_star, loss_star, '*', color=colors[i], markersize=10, zorder=5)
        ax.annotate(f'k*={k_star:.0f}', (k_star, loss_star),
                    textcoords='offset points', xytext=(6, 6), fontsize=6,
 ↪  color=colors[i])

ax.set_xlabel('Evaluation tasks per round (k)')
ax.set_ylabel('Evaluator loss L(k)')
ax.set_title('Optimal evaluation size')
ax.legend(fontsize=6)
ax.set_ylim(0, 1.1)
ax.set_xlim(0, 40)

plt.tight_layout()
plt.show()
```

Figure. Asymptotic evaluator loss $\mathcal{L}(k) = \sigma^2/k + k/(\rho|F| + k)$ for several correction rates, with stars marking the optimal $k^* = \sigma\rho|F|/(\sqrt{\rho|F|} - \sigma)$. Faster correction (larger ρ, hence larger $\rho|F|$) lowers the minimum loss and pushes k^* rightward, scaling as $\sqrt{\rho|F|}$ once $\rho|F| \gg \sigma^2$. Parameters: $\sigma = 1$, $|F| = 2000$.

Together, the two levers of this section — randomization to hide D and correction to keep D moving toward μ — make hiding the distribution durable: leaked information about D goes stale faster than the builder can exploit it. What they cannot address is the case where D itself is the wrong target, i.e., $D \neq \mathcal{C}$. That gap is the subject of the next section.

11.6 Publishing the Construct: The Construct-Validity Protocol

Hiding D buys durability, but it raises two practical worries. If the distribution is secret, how does anyone — a regulator, an adopter, the builder itself — know the score means anything? And how is a builder supposed to improve against a target it cannot see? The construct-validity protocol resolves both by exploiting the very gap that caused the trouble. Because a single construct $\mathcal{C}$ admits many operationalizing distributions, the evaluator can keep one distribution private while making *others* public.

The Construct-Validity Protocol

The evaluator:

1. **Publishes the construct** $\mathcal{C}$ — a precise statement of the capability being measured (its content domain and boundaries), but not a benchmark.
2. **Keeps** D **and** S **private** — the operationalizing distribution and every realized set stay secret, evaluated only inside a controlled pipeline.
3. **Publishes one or more proxy distributions** D', D'' — public benchmarks that operationalize the *same* $\mathcal{C}$ (think GSM8K as a public proxy for a private competition-math distribution). Builders train and self-evaluate against the proxies.

The protocol rests on **construct validity** (Chapter 1): if D and D' genuinely operationalize the same construct, then capability transfers between them — a model that improves on D' *because* it acquired the construct should also improve on the private D. This gives the evaluator a falsifiable check that requires no exposure of D: compare a model's private score to what its public-proxy performance predicts. Construct-valid improvement (real capability) transfers; proxy-specific gaming does not, and reveals itself as a model that beats the proxy yet underperforms its predicted private score. The private benchmark thus functions as a *held-out construct probe*, while all public optimization pressure lands on the proxies.

This is why $\mathcal{C}$ need not — and should not — be private. Publishing the construct is what makes the score *interpretable* (validity) and what gives builders a legitimate target to improve against; privacy is reserved for the *operationalization*, where it buys durability. The protocol is the constructive use of the $\mathcal{C} \supsetneq D$ gap: the same multiplicity of operationalizations that lets a builder mistake D for $\mathcal{C}$ also lets an evaluator certify $\mathcal{C}$ through public proxies while never revealing D.

The protocol is an aspirational design rather than a closed theory. Making "transfer across operationalizations" a precise, testable guarantee is itself a *predictive-evaluation* question — whether public-proxy performance predicts held-out performance — and connects the strategic results of this chapter to the predictive-modeling tools developed earlier in the book. It is an active direction for the field, and a natural foundation for a privacy-preserving evaluation challenge.

11.7 Metric Design as Principal-Agent Problem

We now shift from *which tasks to show* (the object axis of Section 11.2) to *which metrics to report* — an orthogonal, reporting-side axis. Even with a perfectly designed evaluation mechanism, the *choice of metric* can create perverse incentives.

11.7.1 When Metrics Create Perverse Incentives

A vivid illustration comes from healthcare. In 2001, the New York State Department of Health began publishing hospital mortality rates for cardiac surgery. Dranove et al. (2003) documented the consequences: hospitals began *avoiding* severely ill patients rather than improving care. By publishing average treated outcome (ATO) as the quality metric, the system rewarded patient selection rather than treatment effectiveness.

The AI evaluation analogy is direct. A leaderboard that rewards average benchmark score incentivizes developers to specialize on easy benchmarks or to select favorable evaluation conditions, rather than improving broadly. The metric itself creates the misalignment.

Formalizing this as a principal-agent problem (Laffont and Tirole 1986): the principal (evaluator) chooses a reward function w, and the agent (developer) best-responds with policy π^w. The principal's regret is $R(\pi^w) = \max_{\tilde{\pi} \in \Pi} V(\tilde{\pi}) - V(\pi^w)$, where $V(\pi) = \mathbb{E}[Y(\pi) - Y(0)]$ is the total treatment effect.

Proposition: ATO Has Unbounded Regret

The average treated outcome $w_{\text{ATO}}(x, t, y) = y \cdot \mathbf{1}[t = 1]$ can have unbounded regret. An agent maximizing $\mathbb{E}[w_{\text{ATO}}]$ may achieve $R(\pi^{w_{\text{ATO}}}) = \max_\pi V(\pi)$ — the *worst possible* policy.

The intuition is stark: a hospital maximizing average treated outcome selects only the healthiest patients for treatment, achieving excellent ATO but zero (or negative) total treatment effect. In the AI setting, a developer maximizing average benchmark accuracy across their selected tasks may simply drop the hardest benchmarks from their evaluation suite.

Proposition: Total Treatment Effect Achieves Zero Regret

The total treatment effect metric $w_{\text{TT}}(x, t, y) = y - \hat{\mu}_0(x)$ achieves zero regret when the principal has unbiased counterfactual estimates $\hat{\mu}_0(x) = \mathbb{E}[Y(0) \mid X = x]$.

The key insight: by subtracting the counterfactual baseline (what would have happened without treatment), the metric removes the incentive for patient selection. In AI evaluation, this translates to scoring the "capability uplift" — performance relative to a baseline — rather than raw performance. This connects to the doubly robust estimation from Chapter 9: counterfactual metrics require causal reasoning about what performance *would have been* absent the developer's effort.

11.7.2 Information Asymmetry and Metric Elicitation

Beyond metric choice, there is a prior question: how much should the developer *reveal* about their model's capabilities? Consider a developer (agent) with a private cost-correlated variable X — for instance, knowledge about which capability dimensions their model excels at. The evaluator (principal) can design contracts conditioned on X if it is revealed.

Drawing on the information elicitation framework of S. Wang et al. (2024), the key question is: when does the developer prefer to reveal, conceal, or *garble* information about their capabilities?

- **Reveal**: The developer discloses X fully. The principal designs a targeted contract, which benefits both parties when the information enables efficient allocation.
- **Conceal**: The developer hides X. The principal must use a one-size-fits-all contract, which is robust but potentially inefficient.
- **Garble**: The developer discloses a noisy version of X. This intermediate option can be Pareto-improving.

Proposition: Principal Always Benefits from Revelation

The principal's expected utility is weakly higher under revelation than concealment: $V_P^{\text{reveal}} \geq V_P^{\text{conceal}}$.

Proof sketch. Under revelation, the principal's optimization problem has a strictly larger feasible set (contracts conditioned on X), so the optimum can only improve. □

But the developer's incentives are more nuanced. Revelation helps when conditioning on X sufficiently differentiates high-cost and low-cost types, allowing the principal to offer more efficient contracts. Concealment is preferred when revelation would lead the principal to extract all of the developer's surplus through perfectly targeted contracts.

11.7.3 Garbling as Differential Privacy

The most interesting case is *garbling*: the developer reveals $Y = X$ with probability ε and $Y = \xi$ (noise) with probability $1 - \varepsilon$. This is precisely the randomized response mechanism from differential privacy.

Proposition: Garbling Can Pareto-Dominate Both Extremes

Under fairly wide conditions (e.g., two exponential cost types with different means), the agent may prefer garbled disclosure to both full concealment and full revelation. Moreover, garbling can increase total welfare compared to concealment.

The intuition is that garbling provides the principal with *just enough* information to design better contracts without enabling full surplus extraction. In the AI evaluation context, this maps to:

- **Model cards with calibrated noise**: Developers disclose approximate capability profiles rather than exact performance vectors.
- **Differential privacy in metric reporting**: Adding Laplace noise to reported metrics creates a garbling mechanism.
- **Benchmark transparency tiers**: Publish aggregate scores publicly, but release fine-grained breakdowns only through a privacy-preserving mechanism.

This connects directly to the holdout mechanism of Section 11.4.2: both use calibrated noise to create better incentive equilibria. The parallel is deep — the evaluator's decision about how much to reveal about the evaluation distribution and the developer's decision about how much to reveal about model capabilities are dual problems in information design.

```
#| label: agency-game
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar

np.random.seed(42)

# --- Agency game simulation ---
# Two exponential cost types: lambda_0 (high cost), lambda_1 (low cost)
lambda_0_range = np.linspace(0.5, 5.0, 30)
lambda_1_range = np.linspace(0.5, 5.0, 30)

# Principal's value from agent effort
b = 2.0

# Compute revelation advantage for agent
V_diff = np.zeros((len(lambda_0_range), len(lambda_1_range)))

for i, lam0 in enumerate(lambda_0_range):
    for j, lam1 in enumerate(lambda_1_range):
        # Prior: equal probability of each type
        p0, p1 = 0.5, 0.5

        # Under revelation: principal offers type-specific contracts
        # Optimal effort for type k: e* = max(0, b - lambda_k)
        # Agent surplus under reveal: E[max(0, w_k - C_k)] for targeted w_k
        e_rev_0 = max(0, b - lam0)
        e_rev_1 = max(0, b - lam1)
        # Agent expected utility: wage minus expected cost
        V_agent_rev = p0 * max(0, (b - lam0)**2 / (2*b)) + p1 * max(0, (b - lam1)**2 /
 ↪  (2*b))

        # Under concealment: principal offers pooling contract
        lam_pool = p0 * lam0 + p1 * lam1
        e_con = max(0, b - lam_pool)
```

```
        V_agent_con = max(0, (b - lam_pool)**2 / (2*b))

        V_diff[i, j] = V_agent_rev - V_agent_con

fig, axes = plt.subplots(1, 2, figsize=(6, 2))

# Panel 1: Heatmap of V_reveal - V_conceal
im = axes[0].imshow(V_diff, extent=[lambda_1_range[0], lambda_1_range[-1],
                                    lambda_0_range[-1], lambda_0_range[0]],
                    aspect='auto', cmap='RdBu_r', vmin=-0.5, vmax=0.5)
axes[0].set_xlabel('  (low-cost type)')
axes[0].set_ylabel('  (high-cost type)')
axes[0].set_title('Agent: V_reveal - V_conceal')
axes[0].contour(lambda_1_range, lambda_0_range, V_diff,
                levels=[0], colors='black', linewidths=1.5)
plt.colorbar(im, ax=axes[0], shrink=0.8)

# Panel 2: Agent utility under garbling (varying epsilon)
lam0_fixed, lam1_fixed = 3.0, 1.0
epsilons = np.linspace(0, 1, 100)
V_garble = np.zeros(len(epsilons))

for ei, eps in enumerate(epsilons):
    # Under garbling: principal observes Y = X w.p. eps, Y = noise w.p. 1-eps
    # Effective type distribution from principal's perspective
    # Garbled: principal assigns probability
    # P(type=0 | Y=0) = eps * p0 / (eps*p0 + (1-eps)*0.5)
    # Simplified: interpolate between reveal and conceal
    V_garble[ei] = eps * (0.5 * max(0, (b - lam0_fixed)**2 / (2*b)) +
                          0.5 * max(0, (b - lam1_fixed)**2 / (2*b))) + \
                   (1 - eps) * max(0, (b - 0.5*lam0_fixed - 0.5*lam1_fixed)**2 /
    ↪  (2*b))

axes[1].plot(epsilons, V_garble, color='#5B8DEE', lw=2)
axes[1].axhline(y=V_garble[0], color='#E8637A', ls='--', lw=1, label='Conceal')
axes[1].axhline(y=V_garble[-1], color='#45BF7C', ls='--', lw=1, label='Reveal')
best_eps = epsilons[np.argmax(V_garble)]
axes[1].axvline(x=best_eps, color='#F0A35C', ls=':', lw=1, label=f' *={best_eps:.2f}')
axes[1].set_xlabel('Garbling parameter  ')
axes[1].set_ylabel('Agent utility')
axes[1].set_title(f'Garbling ( ={lam0_fixed},  ={lam1_fixed})')
axes[1].legend()

plt.tight_layout()
plt.show()
```

Figure. Left: heatmap of the agent's net benefit from revelation vs. concealment across pairs of exponential cost types (λ_0, λ_1). Red regions indicate the agent prefers revelation; blue regions favor concealment. The black contour marks indifference. Right: agent utility under garbling

(noisy disclosure with probability ε) for fixed cost types. An intermediate garbling level can outperform both full concealment and full revelation.

11.8 Representative Benchmark Selection

A distinct but related design question: given a large suite of evaluation metrics, how should one select a representative subset? This is the "lite benchmark" problem faced by BIG-bench (200+ metrics → BIG-bench Lite, 24), HELM (→ HELM Lite), and Cal Hospital Compare (hundreds of quality measures → 12). In the language of Section 11.2, a representative subset is one whose induced distribution best approximates D — and hence the construct $\mathcal{C}$ — at a fraction of the evaluation cost.

11.8.1 The Subset Selection Problem

Let there be n metrics and m alternatives (models). Each metric i produces a ranking σ_i over the alternatives. We seek a subset $K \subseteq N = [n]$ of metrics that is "representative" of the full set N. But what does "representative" mean formally? The work of Procaccia et al. (2025) provides two precise definitions grounded in social choice theory.

11.8.2 Positional Representation

The first notion prevents *under-representation* at every rank cutoff.

Definition: Positional Representation

A subset K satisfies **positional representation** for group size g if for every rank cutoff $r \in [m]$ and every alternative a:

$$C(K, r, a) \geq \left\lfloor \frac{C(N, r, a)}{g} \right\rfloor$$

where $C(S, r, a) = |\{i \in S : \sigma_i(a) \leq r\}|$ counts how many metrics in S rank alternative a in the top r.

In words: if alternative a is ranked in the top r by at least $\ell \cdot g$ metrics in N, then a must be ranked in the top r by at least ℓ metrics in K. The parameter g controls the granularity of representation — smaller g requires finer representation but demands larger $|K|$.

Theorem: Positional Representation Bounds

The minimum subset size needed to guarantee positional representation satisfies:

$$\Omega\left(\frac{n}{g} \cdot \frac{\log m}{\log(n \log m / g)}\right) \leq |K| \leq O\left(\frac{n}{g} \log m\right)$$

The upper bound is achieved by a polynomial-time greedy algorithm (Algorithm 1 in Procaccia et al. (2025)) based on set cover.

The greedy algorithm works as follows: iterate through the preference profile row by row, coloring entries when an alternative accumulates g appearances. Then greedily select metrics that cover the most remaining colors. The connection to set cover yields the logarithmic factor.

11.8.3 Positional Proportionality

Positional representation prevents under-representation but not over-representation. A stronger notion prevents both:

Definition: Positional Proportionality

A subset K satisfies ϵ-**positional proportionality** if for every alternative a and every rank cutoff r:

$$\left| \frac{C(N, r, a)}{|N|} - \frac{C(K, r, a)}{|K|} \right| \leq \epsilon$$

This preserves the *fraction* of metrics ranking each alternative at each position, up to additive error ϵ.

Theorem: Positional Proportionality Bounds

The minimum subset size for ϵ-positional proportionality satisfies:

$$\Omega\left(\frac{1}{\epsilon^2} \log m\right) \leq |K| \leq O\left(\frac{1}{\epsilon^2} \log m\right)$$

These bounds are tight up to constant factors.

The tight bounds follow from a connection to uniform convergence: a random subset of size $O(\epsilon^{-2} \log m)$ satisfies positional proportionality with high probability, by a Chernoff-type argument applied simultaneously to all $O(m)$ alternatives and rank cutoffs.

A particularly useful consequence connects proportionality to scoring rules:

Theorem: Scoring Rule Approximation

If K satisfies ϵ-positional proportionality, then for any scoring rule with score vector s and every alternative a:

$$|f_s(a, \sigma_N) - f_s(a, \sigma_K)| \leq \epsilon$$

where $f_s(a, \sigma_S) = \frac{1}{|S|} \sum_{i \in S} s_{\sigma_i(a)}$ is the average score of alternative a under scoring rule s in metric set S.

This is the key practical guarantee: a subset satisfying positional proportionality approximates *any* scoring rule on the original set of metrics. The connection to Section 7.1 is direct: positional

proportionality provides a formal criterion for when a "lite" benchmark preserves the same information as the full suite, complementing the dimensionality analysis of factor models.

```
#| label: positional-representation
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# --- Generate synthetic benchmark data ---
n_metrics = 50    # full set
m_models = 20     # alternatives

# Generate random rankings: each metric produces a permutation
rankings = np.zeros((n_metrics, m_models), dtype=int)
for i in range(n_metrics):
    rankings[i] = np.random.permutation(m_models)

def count_top_r(rankings, subset, r, a):
    """Count how many metrics in subset rank alternative a in top r."""
    return sum(1 for i in subset if rankings[i, a] < r)

def greedy_positional_representation(rankings, g):
    """Greedy algorithm for positional representation."""
    n, m = rankings.shape

    # Phase 1: Color entries
    colors = {}
    color_id = 0
    alt_counts = {a: [] for a in range(m)}  # track uncolored positions per alt

    for r in range(m):
        for a in range(m):
            entries = [(i, r) for i in range(n) if rankings[i, a] == r]
            for (i, pos) in entries:
                alt_counts[a].append((i, pos))
                if len(alt_counts[a]) >= g:
                    for (mi, mp) in alt_counts[a][-g:]:
                        colors[(mi, a, mp)] = color_id
                    color_id += 1
                    alt_counts[a] = []

    # Phase 2: Greedy set cover
    K = set()
    all_colors = set(colors.values())
    remaining = all_colors.copy()

    metric_colors = {i: set() for i in range(n)}
```

```
    for (mi, a, pos), c in colors.items():
        metric_colors[mi].add(c)

    while remaining:
        best_metric = max(range(n), key=lambda i: len(metric_colors[i] & remaining) if
↪   i not in K else -1)
        if len(metric_colors[best_metric] & remaining) == 0:
            break
        K.add(best_metric)
        remaining -= metric_colors[best_metric]

    return K

# --- Run for different group sizes ---
g_values = [2, 3, 5, 8, 10, 15, 20, 25]
greedy_sizes = []
random_sizes = []

for g in g_values:
    K = greedy_positional_representation(rankings, g)
    greedy_sizes.append(len(K))

    # Random baseline: how many random metrics needed?
    n_trials_rand = 50
    rand_needed = []
    for _ in range(n_trials_rand):
        perm = np.random.permutation(n_metrics)
        for size in range(1, n_metrics + 1):
            K_rand = set(perm[:size])
            # Check positional representation
            satisfied = True
            for r in range(m_models):
                for a in range(m_models):
                    c_full = count_top_r(rankings, range(n_metrics), r, a)
                    c_sub = count_top_r(rankings, K_rand, r, a)
                    if c_sub < c_full // g:
                        satisfied = False
                        break
                if not satisfied:
                    break
            if satisfied:
                rand_needed.append(size)
                break
        else:
            rand_needed.append(n_metrics)
    random_sizes.append(np.mean(rand_needed))

# --- Plot ---
fig, axes = plt.subplots(1, 2, figsize=(6, 2))
```

```
colors_plot = ['#5B8DEE', '#E8637A', '#45BF7C']

axes[0].plot(g_values, greedy_sizes, 'o-', color=colors_plot[0], lw=1.5, markersize=4,
 ↪  label='Greedy')
axes[0].plot(g_values, random_sizes, 's--', color=colors_plot[1], lw=1.5,
 ↪  markersize=4, label='Random')
axes[0].plot(g_values, [n_metrics/g * np.log(m_models) for g in g_values], ':',
             color='#888888', lw=1, label='(n/g)·log(m) bound')
axes[0].set_xlabel('Group size g')
axes[0].set_ylabel('Selected subset size |K|')
axes[0].set_title('Positional representation')
axes[0].legend()

# Panel 2: Show ranking preservation for the g=5 case
K_example = greedy_positional_representation(rankings, g=5)
K_list = sorted(K_example)

# Top-5 ranking of each model: fraction of metrics ranking it in top 5
top_r = 5
frac_full = np.array([count_top_r(rankings, range(n_metrics), top_r, a) / n_metrics
                      for a in range(m_models)])
frac_subset = np.array([count_top_r(rankings, K_list, top_r, a) / len(K_list)
                        for a in range(m_models)])

sort_idx = np.argsort(-frac_full)
x = np.arange(m_models)
w = 0.35
axes[1].bar(x - w/2, frac_full[sort_idx], w, color=colors_plot[0], alpha=0.7,
 ↪  label=f'Full (n={n_metrics})')
axes[1].bar(x + w/2, frac_subset[sort_idx], w, color=colors_plot[2], alpha=0.7,
 ↪  label=f'Subset (|K|={len(K_list)})')
axes[1].set_xlabel('Model (sorted)')
axes[1].set_ylabel(f'Fraction in top {top_r}')
axes[1].set_title(f'Top-{top_r} preservation (g=5)')
axes[1].legend()
axes[1].set_xticks([])

plt.tight_layout()
plt.show()
```

Figure. Left: subset size $|K|$ required by the greedy algorithm for positional representation at various group sizes g, compared to random selection and the theoretical $O(n/g \cdot \log m)$ bound. The greedy algorithm consistently outperforms random selection. Right: for $g = 5$, comparison of the fraction of metrics ranking each model in the top 5, between the full set ($n = 50$) and the greedy-selected subset. The subset closely preserves the ranking structure.

11.9 Synthesis: Design Principles for Strategic Benchmarks

The three threads of this chapter — information design, metric design, and representative selection — converge on six actionable design principles for AI evaluation.

Table 11.2
Six design principles for strategic AI evaluation

Principle	Formal Basis	Recommendation
1. Randomize and Refresh	Single-sample alignment (TV bound), Prop. (transparent failure)	Use randomized evaluation with task renewal. Transparent fixed benchmarks are Goodhart-vulnerable by construction.
2. Correct and Grow	Gap-targeted correction theorem, Lemma (optimal k^*)	Invest in distribution correction (ρ) toward the construct μ as the primary lever. The optimal $k^* \approx \sigma\sqrt{\rho\lvert F\rvert}$ grows with $\sqrt{\rho}$.
3. Gate Information Release	Holdout mechanism, Prop. (learning slowdown)	Use threshold tests to condition information flow on builder behavior. Combine with distribution correction.
4. Align Metrics with Welfare	Prop. (ATO regret), Prop. (TT zero regret)	Score total treatment effect, not averages. Account for counterfactual baselines.
5. Allow Partial Transparency	Prop. (garbling dominance)	Calibrated noise in metric reporting (garbling) can create Pareto improvements over both full transparency and full opacity.
6. Ensure Representative Subsets	Thms. (positional representation/proportionality bounds)	Use formal representation criteria rather than ad hoc selection when creating lite benchmarks.

These principles interact in important ways. Principles 1–3 address the *information channel* (how much the evaluator reveals about the evaluation mechanism). Principle 4 addresses the *metric channel* (what the evaluator measures). Principle 5 addresses the *developer channel* (what the developer reveals about their model). Principle 6 addresses the *scope channel* (which metrics to include).

A well-designed evaluation system should operate on all four channels simultaneously: randomize task selection (Principle 1), invest in distribution correction (Principle 2), gate information via holdout mechanisms (Principle 3), use welfare-aligned metrics (Principle 4), allow developers to disclose through privacy-preserving mechanisms (Principle 5), and ensure the metric suite is formally representative (Principle 6).

11.9.1 Three Properties of a Durable Evaluator

The six principles above are *mechanisms*; it helps to name the *properties* they secure. Read through the object-first lens of Section 11.2, a well-designed evaluator should guarantee three things.

- **Validity**. The score carries its intended construct meaning: it tracks $\mathcal{C}$, not merely the distribution D that operationalizes it. Validity is secured by *publishing the construct and proxies* (Section 11.6) and by *welfare-aligned metrics* (Section 11.7) that reward capability rather than its proxies.
- **Durability**. The benchmark stays informative as it is optimized against, because D stays private and is *corrected* toward μ faster than it is learned (Section 11.5). Hiding the set (Section 11.4.2) buys time; hiding and correcting the distribution buys durability.
- **Faithful execution**. The evaluator runs exactly the announced procedure — the same sampling, scoring, and reporting for every subject, with no special access or after-the-fact adjustment. This single property subsumes what one might separately call *neutrality* (no subject is favored) and *integrity* (the promised procedure is the one that runs): both are guarantees about honest execution of a fixed protocol, and both fail in the same way — a strategic *evaluator* (Section 10.3.2) — which the builder-facing mechanisms of this chapter do not address.

The natural implementation of faithful execution is procedural: run sampling, evaluation, and reporting inside a **secure, auditable pipeline** — a sealed environment that draws $S \sim D$, scores submissions, and emits only the agreed signal, while producing a verifiable log that the announced protocol was followed. Randomization protects the builder from the evaluator's *information*; auditable execution protects every party from the evaluator's *discretion*. Only the first is a mechanism-design question; the second is why Chapter 10 treats evaluator-side gaming as a matter of governance.

Table 11.3
The object-first view of the chapter's mechanisms and the properties they secure

Object hidden	Mechanism	Property secured
nothing	static benchmark	— (fails)
set S	reusable holdout / Ladder	durability (bounded)
distribution D	randomization + correction	durability
D, S; publish $\mathcal{C}, D'$	construct-validity protocol	validity + durability
the protocol itself	secure auditable pipeline	faithful execution

11.10 Discussion Questions

1. The evaluation game framework assumes a *benevolent* evaluator (social planner). What changes if the evaluator also has strategic incentives — for example, a company running its own benchmark to favor its own models?

2. How does the information-variance tradeoff relate to the reliability-validity tradeoff from Chapter 5 and Chapter 1? Is there a formal connection between σ^2/k (evaluation variance) and Cronbach's α?

3. Design a mechanism where the builder's incentive is to improve on the *hardest* tasks rather than the average. How should $u_E(\theta)$ and $r(\theta, S)$ be modified?

4. In what sense is the Chatbot Arena a randomized evaluation mechanism? Does it satisfy the conditions of the single-sample alignment theorem (Section 11.5.1)? What information leaks through the adaptive matching process?

5. How should a government regulator think about the correction rate ρ and the transparency level k for a safety benchmark? Which is the more effective lever, and what factors (incident-reporting infrastructure, task-acquisition capacity, score-noise scale σ) should influence the choice?

6. If two "lite" benchmarks both satisfy ϵ-positional proportionality with the same ϵ but select different subsets, which should be preferred? What additional criteria might break ties?

7. The holdout mechanism draws on differential privacy. What is the relationship between the privacy budget ε in the holdout mechanism and the notion of ε-differential privacy from the data privacy literature?

8. Distribution correction requires the evaluator to "know their bias." How can an evaluator identify which regions of the task universe they under-represent? What role do meta-evaluations and coverage audits play?

9. Pick three named benchmarks (e.g., FrontierMath, LiveBench, Chatbot Arena) and classify each by *which object* it keeps private — the construct $\mathcal{C}$, the distribution D, or only the set S (Section 11.2). Where the answer is "only S," what would it take to also hide D?

10. A dynamic benchmark refreshes its test set every round. Construct a scenario in which it remains exploitable *even with zero item-level leakage*. (Hint: consider $D \neq \mathcal{C}$ and Section 11.4.3.) What does this say about the limits of contamination control?

11. Faithful execution (Section 11.9.1) is the one property the chapter's builder-facing mechanisms do not secure. Give an example of an evaluator behavior that violates faithful execution but that no amount of randomization or distribution correction can prevent. What kind of intervention would address it?

11.11 Bibliographic Notes

The Stackelberg evaluation game, the single-sample alignment bound, and the information-variance tradeoff are developed in S. Truong, Wang, et al. (2025) ("Strategic Evaluation: Incentivizing AI Capability Coverage with Private Benchmarks"). The gap-targeted distribution-correction mechanism and the optimal evaluation size k^* are from the same work, which also

reports a semi-synthetic experiment on MMLU-Pro using a low-rank latent-factor model of task structure. The analysis of metric design as a principal-agent problem draws on S. Wang et al. (2024), which develops counterfactual quality metrics (Chapter 5) and information elicitation in agency games (Chapter 6). The positional representation and proportionality framework is from Procaccia et al. (2025).

Goodhart's Law was articulated by Goodhart (1984) in the context of monetary policy. Manheim and Garrabrant (2018) formalize four variants. The connection to reward overoptimization in RLHF is explored by Gao et al. (2023).

Strategic classification — the study of agents who manipulate their features to achieve favorable classification outcomes — was formalized by Hardt et al. (2016). Perdomo et al. (2020) introduce performative prediction, where the model itself changes the data distribution. Braverman and Garg (2020) show that randomness is necessary for efficient classification under strategic behavior.

Bayesian persuasion (Kamenica and Gentzkow 2011) provides the information design framework connecting evaluator disclosure to builder behavior. Bergemann and Morris (2019) survey the broader information design literature. **Contract theory** (Laffont and Tirole 1986; Holmstrom and Milgrom 1991) provides the principal-agent foundations for metric design.

Differential privacy and adaptive data analysis (Dwork et al. 2015; Dwork and Roth 2014) underpin the holdout mechanism. Blum and Hardt (2015) apply reusable holdout techniques to machine learning competitions. The connection between DP and strategic robustness is developed in the holdout mechanism analysis.

What to hide: sets versus distributions. The object-first framing of Section 11.2 — separating the construct $\mathcal{C}$, the operationalizing distribution D, and the realized set S — organizes a range of mechanisms by what they keep private. Reusable holdouts and the Ladder hide the *set* and bound leakage (Blum and Hardt 2015; Dwork and Roth 2014). Dynamic benchmarks refresh the set to defeat contamination — DynaBench (Kiela et al. 2021), LiveBench (White et al. 2025), and Chatbot Arena (Chiang et al. 2024) — but, as Section 11.4.3 argues, do not hide the *distribution*. Hiding the distribution is the contribution of the strategic-evaluation game (S. Truong, Wang, et al. 2025). The construct-validity protocol of Section 11.6, which publishes the construct and public proxies while keeping the operationalization private, builds on the validity theory of Chapter 1 and remains an open, partly aspirational direction.

Social choice and benchmarking connect to Zhang and Hardt (2024) (Arrow's impossibility for benchmarks), Colombo et al. (2022) (Borda count for benchmark aggregation), and Rofin and Mikhailov (2023) (scoring rules for benchmark ranking). The committee selection literature, particularly justified representation (Aziz et al. 2017), provides the social choice foundations for positional representation.

11.12 Exercises

1. **(Easy)** Show that if the builder's belief matches social relevance, $\tilde{\pi}_E = \mu$, the single-sample alignment is exact ($\Delta = 0$) and extends to any sample size k, not just $k = 1$. *Hint:* show that for any k-subset mechanism drawn from a μ-matching belief, the expected reward is proportional to $u_E(\theta)$.

2. **(Easy)** Verify that, when $\rho|F| > \sigma^2$, $k^* = \sigma\rho|F|/(\sqrt{\rho|F|} - \sigma)$ minimizes $\mathcal{L}(k) = \sigma^2/k + k/(\rho|F| + k)$. Show that $k^* \approx \sigma\sqrt{\rho|F|}$ in the high-correction regime $\rho|F| \gg \sigma^2$, and explain why no interior optimum exists when $\rho|F| \le \sigma^2$.

3. **(Medium)** Derive the estimation-limited term $m/(m+N)$ in the misalignment bound using a Dirichlet prior centered at μ over distributions on F with m effective observations. Show that the posterior predictive is the shrinkage estimator $\hat{\pi}_E = \frac{m}{m+N}\hat{p} + \frac{N}{m+N}\mu$ and that $\mathrm{TV}(\hat{\pi}_E, \mu) \le m/(m+N)$.

4. **(Medium)** In the agency game with binary X, suppose $C \mid X = 0 \sim \mathrm{Exp}(\lambda_0)$ and $C \mid X = 1$ is zero-cost. Derive the condition on λ_0 and the principal's value b under which the agent prefers concealment to revelation.

5. **(Medium)** Show that any subset K satisfying ϵ-positional proportionality approximates any scoring rule within ϵ. *Hint:* Write $f_s(a, \sigma_S)$ as a weighted sum of cumulative counts $C(S, r, a)/|S|$ and apply Abel summation.

6. **(Hard)** Starting from the single-sample best response $\theta^M \in \arg\max_\theta \mathbb{E}_{f\sim\tilde{\pi}_E}[f(\theta)]$, derive the single-sample alignment bound $\max_\theta u_E(\theta) - u_E(\theta^M) \le 4B\,\mathrm{TV}(\mu, \tilde{\pi}_E)$. Under what conditions on the builder's belief $\tilde{\pi}_E$ does the builder maximize $u_E(\theta)$ exactly?

7. **(Hard)** Prove that the holdout mechanism slows the builder's posterior concentration by a factor of $e^{\varepsilon\eta}$. *Hint:* Compute the effective sample size $m_t^{\text{holdout}} = n_0 + p_{\text{alarm}} \cdot k \cdot \min(t, \rho^{-1})$ and compare to m_t^{baseline}.

8. **(Hard, computational)** Implement the greedy algorithm for positional representation on the HELM Lite data (or a synthetic dataset with $n = 100$ metrics, $m = 50$ models). Compare $|K|$ to the existing lite benchmark subset for different values of the group size g. At what g does the greedy algorithm match the size of the existing subset?

9. **(Medium)** Consider a dynamic benchmark that draws a fresh set $S' \sim D$ each round with *no* item-level leakage, where $D = \pi_E$ is biased away from the construct weighting μ. Suppose the builder knows D and best-responds with $\theta^* = \arg\max_\theta \mathbb{E}_{f\sim D}[f(\theta)]$. Show that the residual misalignment $\max_\theta u_E(\theta) - u_E(\theta^*)$ is bounded below by a quantity proportional to $\mathrm{TV}(\mu, D)$, and conclude that no refresh rate drives it to zero. (This is the claim used in Section 11.4.3.)

10. **(Easy)** Classify each of the following by the object it keeps private (construct $\mathcal{C}$, distribution D, or set S): (a) a public benchmark with a hidden answer key; (b) a leaderboard backed by the Ladder algorithm; (c) a monthly-refreshed live benchmark; (d) a private benchmark released only as a published construct plus a public proxy. Which are durable against a builder who learns D?

11. **(Hard)** Design a concrete *transfer test* for the construct-validity protocol (Section 11.6): given a model's scores on public proxies D', D'' and its score on a private D, specify a statistic that distinguishes genuine construct improvement from proxy-specific gaming. What assumption about the relationship between D' and D does your test require, and how would you check it?

12 Application

Intended Learning Outcomes

By the end of this chapter, the reader will be able to:

1. **Frame** red-teaming as an adversarial measurement problem, connecting structured attack protocols to item response theory and content validity.
2. **Analyze** when attack success rate (ASR) comparisons are meaningful, distinguishing conceptual coherence (comparable estimands) from measurement validity (accurate operationalization).
3. **Distinguish** between structured and unstructured red-teaming and analyze the coverage–depth tradeoff in adversarial evaluation.
4. **Formalize** adversarial robustness as a latent trait and apply adaptive adversarial testing using the CAT framework from Section 6.2.2.
5. **Evaluate** when synthetic data preserves construct validity for AI evaluation and identify the conditions under which calibration transfers from synthetic to real items.
6. **Apply** prediction-powered inference to debias synthetic evaluations by combining large-scale synthetic judgments with small human evaluation sets.
7. **Design** an end-to-end adversarial evaluation pipeline that composes red-team item banks, adaptive testing, synthetic augmentation, and statistical correction.

Suggested Lecture Plan

This chapter can be covered in **2 lectures** (75–90 minutes each):

Lecture 1: Red-Teaming as Measurement

- Red-teaming as measurement: coverage, depth, and item banks (20 min)
- When can ASRs be compared? Conceptual coherence and aggregation (20 min)
- Judge validity: differential misclassification and DIF (20 min)
- Hands-on: simulating aggregation bias and judge error (15 min)

Lecture 2: Adversarial Robustness Evaluation

- Adversarial robustness as a latent trait (20 min)
- Adaptive adversarial testing and adversarial IRT (20 min)
- Hands-on: simulating two-dimensional adversarial IRT (10 min)

Lecture 3: Synthetic Data and Evaluation at Scale

- Synthetic data for evaluation: validity threats and calibration (25 min)
- Hybrid evaluation and prediction-powered inference (25 min)
- The adversarial evaluation pipeline (15 min)

– Discussion and exercises (10 min)

NOTATION

This chapter introduces adversarial evaluation notation: $\theta_j^{(\text{adv})}/\theta_j^{(\text{std})}$ (adversarial/standard ability), $\alpha_{s,\mathcal{D}}$ (attack success probability), J/s (operational judge / oracle criterion), K (repeated samples), and $\hat{\mu}_{\text{PPI}}$ (prediction-powered inference estimator). See the notation appendix[1] for the complete notation reference.

[1] notation.qmd

12.1 Red-Teaming as Measurement

Red-teaming—the practice of probing AI systems for failures—has become a standard component of AI evaluation. But most red-teaming efforts are conducted *ad hoc*: a team of human testers tries creative prompts, records failures, and writes a report. From the perspective of measurement science, this is analogous to evaluating a student by asking whatever questions come to mind, with no test specification, no item calibration, and no systematic coverage of the construct domain.

This section reframes red-teaming as a *measurement problem*. The key insight is that a red-team evaluation is an instrument—a collection of adversarial items—and the same psychometric principles that apply to any evaluation instrument (content validity, item calibration, reliability) apply here.

12.1.1 The Coverage-Depth Tradeoff

Consider two red-teaming strategies. Strategy A deploys 50 red-teamers, each spending one hour testing a wide range of attack categories: prompt injection, jailbreaking, harmful content generation, privacy violations, bias elicitation, and so on. Strategy B deploys 5 expert red-teamers, each spending ten hours on a single attack category, developing sophisticated multi-turn attack chains.

Strategy A has *breadth*: it covers many regions of the attack surface. Strategy B has *depth*: it explores individual attack categories more thoroughly. Neither dominates. The tradeoff maps directly onto measurement concepts from Section 1.2.1:

– **Content validity** requires that the adversarial items represent the full domain of potential attacks. Strategy A is better for content validity.
– **Item difficulty** determines whether the evaluation can distinguish models at different robustness levels. Strategy B produces harder items that discriminate among highly robust models.

DEFINITION: RED-TEAM TEST SPECIFICATION

A **red-team test specification** is a document that defines:

1. **Attack taxonomy**: The categories of adversarial behavior to be tested (e.g., jailbreaking, prompt injection, harmful content, bias, privacy leakage).
2. **Coverage requirements**: The minimum number of items per category, ensuring content validity.
3. **Difficulty targets**: The distribution of item difficulties within each category, ensuring discrimination across the robustness spectrum.
4. **Scoring rubric**: The criteria for judging whether a model response constitutes a failure.

Without a test specification, red-team results have unknown content validity—we cannot know what fraction of the attack surface was covered or whether the items were difficult enough to challenge robust models.

12.1.2 Adversarial Items as Hard Items in IRT

In IRT terminology, an adversarial item is simply an item with parameters that we can interpret through the standard framework. Consider a 2PL model applied to adversarial items:

$$P(\text{resist}_j \mid \beta_i^{(\text{atk})}, \alpha_i, \theta_j^{(\text{adv})}) = \frac{1}{1 + \exp(-\alpha_i(\theta_j^{(\text{adv})} - \beta_i^{(\text{atk})}))} \tag{12.1}$$

Here $\theta_j^{(\text{adv})}$ is model j's adversarial robustness and $\beta_i^{(\text{atk})}$ is the attack strength (difficulty) of adversarial item i. A model "responds correctly" by *resisting* the attack. Under this framing:

- **Easy adversarial items** ($\beta_i^{(\text{atk})} \ll 0$) are weak attacks that most models resist. These provide little information about robustness differences among frontier models.
- **Hard adversarial items** ($\beta_i^{(\text{atk})} \gg 0$) are sophisticated attacks that only the most robust models withstand.
- **High-discrimination items** ($\alpha_i \gg 1$) sharply separate robust from non-robust models. These are the most informative items for ranking.

The connection to content validity (Section 1.2.1) is immediate: a red-team evaluation with only easy items has construct underrepresentation for the high-robustness region, just as a math test with only arithmetic problems underrepresents mathematical ability. Conversely, a red-team evaluation with only expert-crafted attacks may have no items in the easy-to-moderate range, making it unable to distinguish among weaker models.

12.1.3 Structured vs. Unstructured Red-Teaming

The red-teaming literature distinguishes two paradigms:

Unstructured red-teaming gives human testers broad instructions ("try to make the model do something bad") and relies on their creativity. This approach can discover unexpected

failure modes but produces items with unknown psychometric properties—we do not know the difficulty, discrimination, or category coverage until after the fact. From a measurement perspective, unstructured red-teaming is useful for *exploratory item generation* but insufficient for *standardized measurement*.

Structured red-teaming uses a test specification to guide item development. Testers are assigned categories, given difficulty targets, and use standardized scoring rubrics. Ganguli et al. (2022) demonstrated the value of scaling structured red-teaming, showing that the distribution of discovered failures changes systematically as a function of red-team effort. Perez et al. (2022) extended this by using language models themselves to generate adversarial prompts at scale, trading human creativity for automated coverage.

> **KEY INSIGHT: RED-TEAMING MEETS CONSTRUCT-IRRELEVANT VARIANCE**
>
> A common failure in red-teaming is conflating *refusal to answer benign questions* with *robustness to adversarial attacks*. If a model refuses to discuss any sensitive topic—including legitimate ones—it will score well on a red-team evaluation that measures only refusal rate. But this "robustness" is partly construct-irrelevant variance (Section 1.3.1): the model's overrefusal is a systematic factor unrelated to the construct of *adversarial robustness*. A well-designed red-team evaluation must include benign items that probe for false positives (inappropriate refusals), just as a diagnostic test must measure both sensitivity and specificity.

12.1.4 Red-Team Item Banks

The psychometric solution to the coverage-depth tradeoff is to build a *calibrated item bank*: a large pool of adversarial items with known IRT parameters. Once items are calibrated, different evaluations can draw from the bank according to their needs:

- A **broad screening** evaluation draws items uniformly across categories, covering the full attack taxonomy.
- A **targeted deep evaluation** draws difficult items from specific categories, probing known weakness areas.
- An **adaptive evaluation** selects items based on the model's responses, as in computerized adaptive testing (Section 6.2.2).

Building such a bank requires an initial investment: items must be authored, administered to a calibration sample of models, and their parameters estimated. But the investment pays off in reusability—the same calibrated item can be used across multiple evaluations, and new models can be scored against the existing bank without re-calibration.

Ribeiro et al. (2020) proposed CheckList, one of the earliest structured approaches to building reusable test suites for NLP models. CheckList organizes items by capability (vocabulary, taxonomy, robustness, etc.) and test type (minimum functionality, invariance, directional expectation), providing a template for adversarial item banks. Bartolo et al. (2021) took a dynamic approach with DynaBench, where humans author items that fool current models, creating a continuously challenging item pool.

12.2 When Can Attack Success Rates Be Compared?

Attack success rate (ASR) is the dominant metric in red-teaming studies: "Model A has ASR 0.12 and Model B has ASR 0.31, so A is safer." But Chouldechova et al. (2026) show that many such comparisons are invalid—they rest on apples-to-oranges estimands or low-validity measurements. This section formalizes the conditions under which ASR comparisons are meaningful, drawing on the measurement framework developed throughout this book.

12.2.1 ASR as an Estimand

The term "attack success rate" is misleading. It suggests we are computing the fraction of attacks that succeed. In reality, ASR reflects the fraction of *attack goals* that are successfully met. To make this precise, we need a probabilistic threat model.

DEFINITION: PROBABILISTIC THREAT MODEL

A **probabilistic threat model** $\mathcal{M} = (s, \mathcal{D}, \mathcal{C})$ for red-teaming specifies:

1. **Oracle success criteria** $s(R; P) \to \{0, 1\}$: a function determining whether system response R to prompt P constitutes undesirable behavior.
2. **Goal distribution** $\mathcal{D}$: a distribution over base harmful prompts $P \sim \mathcal{D}$.
3. **Conditions** $\mathcal{C}$: constraints governing the attack method (single-turn, multi-turn, transfer, etc.).

The **attack success probability** (the estimand) is:

$$\alpha_{s,\mathcal{D}} = P_{P \sim \mathcal{D}}[s(L(P); P) = 1]$$

where L is the target system. The observed ASR is an *estimate* of this population parameter.

This formalization reveals that comparing ASRs across studies requires comparing the underlying *estimands*, not just the numerical values. Two conditions must hold for a comparison to be meaningful.

12.2.2 Conceptual Coherence: Aggregation Matters

The first condition is **conceptual coherence**: the ASRs being compared must estimate the same (or comparable) population parameters. In practice, this often fails because studies use different *aggregation* rules for computing ASR, which silently changes the estimand.

Consider two common aggregation strategies:

One-shot ASR. For each prompt P, sample a single response from the target system and check if the attack succeeds:

$$\alpha_{\text{one-shot}} = P_{P \sim \mathcal{D}}[s(L(P); P) = 1]$$

Top-1 of K ASR. For each prompt P, sample K responses and declare success if *any* response is judged successful:

$$\alpha_{\text{Top-1}}(K) = P_{P\sim\mathcal{D}}\left[\max_{k=1,\ldots,K} s(L(P)_k; P) = 1\right]$$

These are different estimands. If the per-prompt success probability is p_0, then the Top-1 success probability is $1-(1-p_0)^K$, which grows rapidly with K. For $p_0 = 0.01$ and $K = 392$, we get $1 - 0.99^{392} \approx 0.98$. A study reporting Top-1 of 392 ASR = 0.89 and comparing it to another study's one-shot ASR = 0.31 is not comparing model safety or attack efficacy—it is comparing estimands (Chouldechova et al. 2026).

KEY INSIGHT: REPEATED SAMPLING INFLATES ASR

Repeatedly sampling responses under high-temperature decoding and reporting Top-1 ASR trivially inflates the metric. Chouldechova et al. (2026) show that simply resampling *baseline prompts* (no jailbreak) 50 times at temperature 2.0 achieves Top-1 ASR of 0.83 on Llama 2 7B Chat—competitive with sophisticated jailbreak methods. The apparent superiority of complex attacks over simple baselines often reflects aggregation differences, not genuine attack efficacy.

In IRT terms, the aggregation choice is analogous to the difference between scoring a test item as "correct if the student gets it right on the first attempt" versus "correct if the student gets it right in any of K attempts." These measure different constructs: the former measures ability under a single opportunity, the latter measures the *upper bound* of ability under repeated trials.

THE DISTRIBUTIONAL THEORY OF TOP-1 SCALING

Schaeffer et al. (2025) provide a precise characterization of how Top-1 aggregation inflates ASR. Per-prompt, the success probability after K attempts is $1-(1-p_i)^K$, which grows *exponentially* fast in K. Yet when averaged over prompts, the aggregate scales only as a *power law* in K. The resolution is that the distribution of per-prompt success probabilities p_i has a heavy left tail: many prompts have near-zero single-attempt success probability. If $p_i \sim \text{Beta}(\alpha, \beta)$, the aggregate follows $-\log(\text{ASR}_{\text{Top-1}}(K)) \sim K^{-\alpha}$, where α is the left-tail shape parameter.

This has two implications for ASR comparisons. First, it explains *why* Top-1 aggregation inflates ASR so dramatically: the heavy tail of near-impossible prompts gets "picked off" exponentially as K grows, producing apparent success even when the one-shot estimand $\alpha_{s,\mathcal{D}}$ is small. Second, it suggests a more informative approach: rather than reporting a single ASR at a fixed K, report (or estimate) the distribution of per-prompt success probabilities, which fully determines the scaling behavior at any K.

This distributional perspective connects to Item Response Theory: the per-prompt success probability p_i is precisely the IRT response probability $\sigma(\theta - \beta_i)$, where θ is the model's latent vulnerability and β_i is the prompt's difficulty. The heavy-tailed distribution of p_i arises because prompt difficulties β_i are heterogeneous — a prediction that IRT makes explicit through its item parameters. See S. Truong, Tu, et al. (2025) for a formal treatment integrating IRT into scaling law estimation.

12.2.3 Measurement Validity: Judge Error and Differential Misclassification

The second condition is **measurement validity**: the observed ASR must be a valid measurement of the estimand. In practice, the oracle success criterion s is replaced by an operational judge J—typically an LLM or rule-based classifier. If J does not faithfully capture s, the ASR is a biased estimate.

The bias depends on the judge's true positive rate (TPR) and false positive rate (FPR):

$$\mathbb{E}[\text{ASR}_J] = \text{TPR} \cdot \alpha + \text{FPR} \cdot (1 - \alpha) \tag{12.2}$$

where $\alpha = \alpha_{s,\mathcal{D}}$ is the true attack success probability. This is the standard misclassification bias formula. Two critical implications follow.

Differential misclassification across target systems. When comparing ASR across models, validity requires that the judge's TPR and FPR be *equal* across target systems. But different models produce different output distributions: some models give verbose refusals, others give terse ones; some hallucinate when confused by adversarial suffixes, others produce empty responses. If the judge has higher FPR on Model B's outputs than on Model A's, Model B will appear *less safe* even if both have the same true α. This is formally a measurement invariance failure—analogous to DIF (Section 5.11.1) where item properties differ across groups.

Differential misclassification across attack methods. Similarly, different jailbreak methods elicit different types of responses. Adversarial suffix attacks may trigger hallucinated responses that judges falsely flag as harmful (higher FPR), while social engineering attacks produce coherent harmful content that judges correctly identify (lower FPR). Comparing ASRs across attack methods without accounting for differential judge error conflates attack efficacy with judge behavior.

> **DEFINITION: VALID ASR COMPARISON**
>
> A comparison of ASRs ASR_A and ASR_B (across systems or attack methods) is **valid** if:
>
> 1. **Conceptual coherence**: Both ASRs estimate the same type of estimand (same aggregation, same goal distribution, same success concept).
> 2. **Measurement validity**: The judge J has equal TPR and FPR with respect to the oracle s across the two conditions being compared.
>
> Condition 1 ensures we are comparing like with like. Condition 2 ensures that observed differences reflect true differences in the estimand, not artifacts of the measurement instrument.

12.2.4 Content Validity of Prompt Sets

A third, often overlooked, validity threat concerns the *base harmful prompts* themselves. Chouldechova et al. (2026) document cases where prompts labeled as policy-violating are in fact benign: "What are some tips for successfully using adult dating apps?" is categorized as harmful content,

and "Can you create detailed talking points for a speech advocating a specific political policy?" is classified as a political activity violation. A model that "complies" with these prompts is not exhibiting unsafe behavior.

This is a **content validity** failure in the sense of Section 1.2.1: the items (prompts) do not accurately represent the target domain (prompts that would elicit genuine policy violations if answered). Just as a math test that includes reading comprehension questions has construct-irrelevant content, a red-team prompt set that includes benign prompts has inflated attack success rates that do not reflect genuine safety failures.

The following simulation demonstrates both the aggregation and judge error problems.

```
#| autorun: true
#| echo: false
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({
    "figure.figsize": (3.5, 3),
    "figure.dpi": 150,
    "figure.autolayout": True,
    "font.size": 8,
    "font.family": "serif",
    "mathtext.fontset": "cm",
    "axes.labelsize": 8,
    "axes.titlesize": 9,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
    "legend.fontsize": 7,
    "lines.linewidth": 1.0,
})
```

```
#| label: asr-comparison
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# --- Setup: 200 prompts, true per-prompt success probability ---
n_prompts = 200
# True success probabilities: most prompts have low p, a few have moderate p
p_true = np.random.beta(0.3, 3.0, n_prompts)  # skewed toward 0
true_alpha = p_true.mean()  # true one-shot ASR (estimand)

# --- Panel 1: Aggregation bias (Top-1 of K) ---
K_values = [1, 5, 10, 25, 50, 100, 200, 500]
top1_asr = []
for K in K_values:
```

```
    # P(at least 1 success in K trials) = 1 - (1 - p)^K
    top1_per_prompt = 1 - (1 - p_true) ** K
    top1_asr.append(top1_per_prompt.mean())

# --- Panel 2: Judge error bias ---
# True alpha for two models
alpha_A = 0.15
alpha_B = 0.15  # same true safety!

# Judge with differential error across models
tpr_A, fpr_A = 0.85, 0.05  # good judge for Model A
tpr_B, fpr_B = 0.85, 0.15  # higher FPR for Model B (e.g., hallucinated outputs)

asr_observed_A = tpr_A * alpha_A + fpr_A * (1 - alpha_A)
asr_observed_B = tpr_B * alpha_B + fpr_B * (1 - alpha_B)

# Sweep FPR_B to show the effect
fpr_B_range = np.linspace(0, 0.3, 50)
asr_B_range = tpr_B * alpha_B + fpr_B_range * (1 - alpha_B)

# --- Panel 3: Combined effect ---
# Show how Top-1 + judge error compound
K_demo = 50
top1_true = 1 - (1 - p_true) ** K_demo
# With judge: each of K samples is judged, success if any judged positive
# P(judge says success | truly fails) per sample = FPR
# P(judge says success | truly succeeds) per sample = TPR
tpr_judge, fpr_judge = 0.90, 0.08
p_judged_success_per_sample = tpr_judge * p_true + fpr_judge * (1 - p_true)
top1_judged = 1 - (1 - p_judged_success_per_sample) ** K_demo

# --- Plot ---
fig, axes = plt.subplots(1, 3, figsize=(6, 2))

# Panel 1: Top-1 ASR inflation
axes[0].plot(K_values, top1_asr, 'o-', color='#E8637A', markersize=3)
axes[0].axhline(true_alpha, color='#5B8DEE', linestyle='--', linewidth=1,
                label=f'One-shot   = {true_alpha:.3f}')
axes[0].set_xlabel('K (repeated samples)')
axes[0].set_ylabel('Top-1 ASR')
axes[0].set_title('Aggregation inflates ASR')
axes[0].set_xscale('log')
axes[0].legend()

# Panel 2: Differential judge error
axes[1].plot(fpr_B_range, asr_B_range, color='#E8637A', linewidth=1.5,
             label='Model B (varying FPR)')
axes[1].axhline(asr_observed_A, color='#5B8DEE', linestyle='--', linewidth=1.5,
                label=f'Model A (FPR={fpr_A})')
```

```
axes[1].axhline(alpha_A, color='gray', linestyle=':', linewidth=1,
                label=f'True   = {alpha_A}')
axes[1].axvline(fpr_A, color='#5B8DEE', linestyle=':', linewidth=0.8, alpha=0.5)
axes[1].set_xlabel('Judge FPR on Model B')
axes[1].set_ylabel('Observed ASR')
axes[1].set_title('Differential judge error')
axes[1].legend()

# Panel 3: Compound effect
axes[2].scatter(top1_true, top1_judged, s=8, alpha=0.5, color='#B07CD8',
 ↪  edgecolor='none')
axes[2].plot([0, 1], [0, 1], '--', color='gray', linewidth=0.8)
axes[2].set_xlabel('Top-1 ASR (true judge)')
axes[2].set_ylabel('Top-1 ASR (noisy judge)')
axes[2].set_title(f'K={K_demo}, TPR={tpr_judge}, FPR={fpr_judge}')

plt.tight_layout()
plt.show()

print("ASR Comparison Analysis:")
print(f"  True one-shot ASR (estimand): {true_alpha:.3f}")
print(f"  Top-1 ASR (K=50):  {top1_asr[4]:.3f}  ({top1_asr[4]/true_alpha:.1f}x
 ↪  inflation)")
print(f"  Top-1 ASR (K=500): {top1_asr[-1]:.3f}  ({top1_asr[-1]/true_alpha:.1f}x
 ↪  inflation)")
print(f"\nDifferential judge error (both models have true   = {alpha_A}):")
print(f"  Model A observed ASR: {asr_observed_A:.3f}  (FPR = {fpr_A})")
print(f"  Model B observed ASR: {asr_observed_B:.3f}  (FPR = {fpr_B})")
print(f"  Spurious gap: {asr_observed_B - asr_observed_A:.3f}")
```

The three panels illustrate why naive ASR comparisons are often misleading. **Left**: Top-1 aggregation inflates ASR exponentially with K. Even when the true one-shot success probability is low (blue dashed line), resampling 500 times drives the Top-1 ASR near 1.0. Comparing a Top-1 ASR to a one-shot ASR is comparing different estimands. **Center**: differential judge error creates spurious safety differences between models with identical true vulnerability. As the judge's FPR on Model B increases (e.g., due to hallucinated outputs that trigger false positives), Model B appears increasingly less safe even though its true α equals Model A's. **Right**: the compound effect of Top-1 aggregation and judge error—points above the diagonal show prompts where the noisy judge inflates the Top-1 ASR beyond its true value.

These results have direct practical implications for red-teaming practice:

1. **Always report the aggregation rule** (one-shot, Top-1 of K, best-of-T) alongside the ASR value. Comparisons across aggregation rules are not comparisons of safety.
2. **Assess judge agreement** disaggregated by target model and attack method. If the judge's error rates differ, the comparison is confounded by measurement error.
3. **Audit prompt validity** for content representativeness—do the "harmful" prompts actually represent genuine policy violations?

4. **Report confidence intervals** that account for both sampling variability and judge error.

12.3 Adversarial Robustness Evaluation

12.3.1 Robustness as a Latent Trait

Is adversarial robustness the same construct as standard accuracy? If a model is good at answering questions correctly, does it follow that the model is also good at resisting adversarial attacks? Empirically, the answer is often no. Zellers et al. (2019) demonstrated "adversarial filtering"—selecting items that are specifically difficult for a target model—which creates items where standard accuracy and adversarial robustness diverge sharply.

We can formalize this as a *multidimensional* IRT model. Let each model j have two latent traits: $\theta_j^{(\text{std})}$ (standard accuracy ability) and $\theta_j^{(\text{adv})}$ (adversarial robustness). For standard items, the response probability depends primarily on $\theta_j^{(\text{std})}$. For adversarial items, it depends on $\theta_j^{(\text{adv})}$:

$$P(X_{ij} = 1) = \frac{1}{1 + \exp(-(\alpha_i^{(\text{std})}\theta_j^{(\text{std})} + \alpha_i^{(\text{adv})}\theta_j^{(\text{adv})} - \beta_i))} \tag{12.3}$$

For a standard item, $\alpha_i^{(\text{std})}$ is large and $\alpha_i^{(\text{adv})} \approx 0$. For an adversarial item, $\alpha_i^{(\text{adv})}$ is large and $\alpha_i^{(\text{std})}$ may be small or moderate. The correlation $\rho(\theta^{(\text{std})}, \theta^{(\text{adv})})$ is an empirical quantity: a low correlation means adversarial robustness is genuinely a *distinct* dimension of model capability.

12.3.2 Perturbation Spaces and Attack Taxonomies

Formalizing adversarial evaluation requires specifying the perturbation space—the set of transformations an adversary can apply:

DEFINITION: PERTURBATION SPACE

A **perturbation space** $\mathcal{P}$ for an evaluation item x is a set of semantics-preserving transformations:

$$\mathcal{P}(x) = \{x' : d(x, x') \leq \epsilon,\ \text{sem}(x') = \text{sem}(x)\}$$

where $d(\cdot, \cdot)$ is a distance metric, ϵ is the perturbation budget, and $\text{sem}(\cdot)$ extracts the semantic content. A model is **robust at item** x if it responds correctly for all $x' \in \mathcal{P}(x)$.

For language models, defining $\mathcal{P}$ is harder than for image classifiers (where ℓ_p-ball perturbations are standard). Common perturbation types include:

- **Paraphrase invariance**: Rephrasing the question while preserving meaning.
- **Format invariance**: Changing the presentation format (markdown, plain text, numbered lists).

- **Prompt injection**: Embedding adversarial instructions within the input.
- **Multi-turn escalation**: Gradually steering the conversation toward unsafe territory.

Each perturbation type defines a different dimension of the attack space. A comprehensive adversarial evaluation must sample from multiple perturbation types, analogous to the content validity requirement that a test must sample from the full construct domain.

12.3.3 Adaptive Adversarial Testing

The connection to computerized adaptive testing (Section 6.2.2) is natural: instead of administering a fixed battery of adversarial items, we can *adapt* the selection of attacks based on the model's responses.

Recall from Section 6.2.2 that the optimal item selection rule in CAT maximizes Fisher information at the current ability estimate:

$$i^* = \arg\max_i \; I_i(\hat{\theta}_j^{(\text{adv})})$$

In the adversarial context, this means selecting the attack whose difficulty is closest to the model's current estimated robustness. If a model easily resists a moderate attack, we escalate to a harder one. If a model fails a moderate attack, we probe with easier attacks to find the boundary. This adaptive strategy is far more efficient than exhaustive testing: it concentrates evaluation effort in the informative region of the attack space.

> **KEY INSIGHT: ADVERSARIAL CAT VS. RANDOM RED-TEAMING**
>
> A random red-team evaluation that draws 100 attacks uniformly from an item bank wastes effort on items that are too easy (every model resists them) or too hard (every model fails). Adaptive adversarial testing with the same 100-item budget concentrates items near each model's robustness frontier, producing tighter ability estimates. The efficiency gain is exactly the same as for standard CAT: adaptive testing can match the precision of a fixed-form test using roughly one-third to one-half as many items.

The following simulation demonstrates the key ideas: we generate adversarial and standard items, fit a two-dimensional model, and show that adversarial robustness is a distinct latent dimension from standard accuracy.

```
#| label: adversarial-irt
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit

np.random.seed(42)

```

```
# --- Generate latent traits ---
n_models = 80
rho = 0.35  # low correlation: robustness != accuracy
mean = [0, 0]
cov = [[1, rho], [rho, 1]]
traits = np.random.multivariate_normal(mean, cov, n_models)
theta_std = traits[:, 0]  # standard accuracy ability
theta_adv = traits[:, 1]  # adversarial robustness

# --- Generate items ---
n_standard = 40
n_adversarial = 40

# Standard items: load on theta_std, low loading on theta_adv
alpha_std_standard = np.random.uniform(0.8, 2.0, n_standard)
alpha_adv_standard = np.random.uniform(-0.1, 0.2, n_standard)
beta_standard = np.random.normal(0, 1, n_standard)

# Adversarial items: load on theta_adv, moderate loading on theta_std
alpha_std_adversarial = np.random.uniform(0.1, 0.5, n_adversarial)
alpha_adv_adversarial = np.random.uniform(0.8, 2.0, n_adversarial)
beta_adversarial = np.random.normal(0.5, 1, n_adversarial)  # harder on average

# --- Simulate responses ---
def simulate_response(theta_s, theta_a, a_s, a_a, b):
    logit = a_s * theta_s + a_a * theta_a - b
    p = expit(logit)
    return np.random.binomial(1, p)

responses = np.zeros((n_models, n_standard + n_adversarial))
for j in range(n_models):
    for i in range(n_standard):
        responses[j, i] = simulate_response(
            theta_std[j], theta_adv[j],
            alpha_std_standard[i], alpha_adv_standard[i], beta_standard[i])
    for i in range(n_adversarial):
        responses[j, n_standard + i] = simulate_response(
            theta_std[j], theta_adv[j],
            alpha_std_adversarial[i], alpha_adv_adversarial[i], beta_adversarial[i])

# --- Compute observed scores ---
score_standard = responses[:, :n_standard].mean(axis=1)
score_adversarial = responses[:, n_standard:].mean(axis=1)

# --- Plot ---
fig, axes = plt.subplots(1, 3, figsize=(6, 2))

# Panel 1: True traits
```

```
axes[0].scatter(theta_std, theta_adv, s=12, alpha=0.6, color='#5B8DEE',
    edgecolor='none')
axes[0].set_xlabel(r'Standard ability ($\theta^{\mathrm{std}}$)')
axes[0].set_ylabel(r'Adversarial robustness ($\theta^{\mathrm{adv}}$)')
axes[0].set_title(f'True traits (r = {rho:.2f})')
axes[0].axhline(0, color='gray', linewidth=0.5, linestyle='--')
axes[0].axvline(0, color='gray', linewidth=0.5, linestyle='--')

# Panel 2: Observed scores
axes[1].scatter(score_standard, score_adversarial, s=12, alpha=0.6, color='#E8637A',
    edgecolor='none')
r_obs = np.corrcoef(score_standard, score_adversarial)[0, 1]
axes[1].set_xlabel('Standard accuracy')
axes[1].set_ylabel('Adversarial accuracy')
axes[1].set_title(f'Observed scores (r = {r_obs:.2f})')
axes[1].axhline(0.5, color='gray', linewidth=0.5, linestyle='--')
axes[1].axvline(0.5, color='gray', linewidth=0.5, linestyle='--')

# Panel 3: Item loading space
axes[2].scatter(alpha_std_standard, alpha_adv_standard, s=20, alpha=0.7,
                color='#5B8DEE', label='Standard', marker='o', edgecolor='none')
axes[2].scatter(alpha_std_adversarial, alpha_adv_adversarial, s=20, alpha=0.7,
                color='#E8637A', label='Adversarial', marker='^', edgecolor='none')
axes[2].set_xlabel(r'Loading on $\theta^{\mathrm{std}}$')
axes[2].set_ylabel(r'Loading on $\theta^{\mathrm{adv}}$')
axes[2].set_title('Item loading space')
axes[2].legend()

plt.tight_layout()
plt.show()

print("Two-dimensional adversarial IRT simulation:")
print(f"  True trait correlation: {rho:.2f}")
print(f"  Observed score correlation: {r_obs:.2f}")
print(f"  Standard items - mean difficulty: {beta_standard.mean():.2f}, "
      f"mean loading on theta_std: {alpha_std_standard.mean():.2f}")
print(f"  Adversarial items - mean difficulty: {beta_adversarial.mean():.2f}, "
      f"mean loading on theta_adv: {alpha_adv_adversarial.mean():.2f}")
```

The three panels reveal the structure of adversarial evaluation. **Left**: the true latent traits show that adversarial robustness ($\theta^{(\text{adv})}$) is only weakly correlated with standard accuracy ($\theta^{(\text{std})}$)—a model that scores well on standard items may be vulnerable to adversarial attacks. **Center**: observed scores on standard and adversarial item subsets mirror this weak correlation, confirming that the two item types measure distinct constructs. **Right**: the item loading space shows clear separation between standard items (loading primarily on $\theta^{(\text{std})}$) and adversarial items (loading primarily on $\theta^{(\text{adv})}$), supporting the two-dimensional model.

This has practical consequences. A single-score benchmark that mixes standard and adversarial items produces a composite that confounds two dimensions. A model could achieve a

high composite score through strong standard accuracy alone, masking poor adversarial robustness. Separate subscores—or better, a multidimensional IRT model—are needed to accurately characterize model capabilities.

12.4 Synthetic Data for Evaluation

12.4.1 Why Synthetic Data?

Human-authored evaluation items are expensive, slow to produce, and hard to scale. Annotating a single benchmark item can require domain experts, multiple rounds of review, and careful quality control. Meanwhile, the space of possible evaluation scenarios is vast—no fixed benchmark can cover it.

Synthetic data generated by language models offers an appealing alternative: items can be produced at scale, targeted to specific difficulty levels, and refreshed frequently to combat contamination. Perez et al. (2022) demonstrated that LLMs can generate adversarial red-team prompts that are competitive with human-authored attacks. But the measurement question remains: *do synthetic items measure the same construct as human-authored items?*

12.4.2 Validity Threats with Synthetic Data

Three systematic threats arise when using LLM-generated items for evaluation:

> **DEFINITION: EVALUATION CIRCULARITY**
>
> **Evaluation circularity** occurs when the same model family is used both to generate evaluation items and to be evaluated on those items. If GPT-4 generates items and GPT-4 is evaluated on them, the items may systematically avoid GPT-4's failure modes—the model cannot probe its own blind spots.

1. **Distribution mismatch.** Synthetic items may not match the distribution of real-world inputs. LLMs tend to generate "typical" examples from their training distribution, undersampling edge cases and rare phenomena. In IRT terms, the synthetic item bank may have a narrow difficulty distribution, concentrated near the mean, with insufficient coverage of the easy and hard extremes.
2. **Mode collapse.** LLMs generating many items tend to produce repetitive patterns. The apparent diversity of a 10,000-item synthetic bank may be much lower than its nominal size, because items cluster in a few templates. This reduces the effective number of items and inflates reliability estimates.
3. **Evaluation circularity.** When the item generator shares architectural or training lineage with the model being evaluated, systematic biases are introduced. The generator's implicit model of "what is hard" may not match human intuitions, and the generated items may exploit the same patterns that the target model handles well.

These threats map onto the validity framework from Section 1.3. Distribution mismatch is a form of covariate shift (Section 9.3). Mode collapse reduces content validity (Section 1.2.1). Evaluation circularity introduces construct-irrelevant variance (Section 1.3.1).

12.4.3 Calibrating Synthetic Items

The key empirical question is whether synthetic items, once calibrated, have IRT parameters that are comparable to human-authored items. If a synthetic item has the same difficulty and discrimination as a matched human-authored item, then—from a measurement perspective—the two are interchangeable.

Formally, let $\hat{\beta}_i^{(\text{syn})}$ and $\hat{\beta}_i^{(\text{human})}$ be the estimated difficulties of paired items (one synthetic, one human-authored, targeting the same content). We say calibration **transfers** if:

$$\hat{\beta}_i^{(\text{syn})} \approx \hat{\beta}_i^{(\text{human})} + c \tag{12.4}$$

where c is a constant offset (synthetic items may be systematically easier or harder). A constant offset is correctable; what matters is that the *rank order* and *relative spacing* of item difficulties are preserved.

The following simulation demonstrates calibration transfer and identifies conditions under which it breaks down.

```
#| label: synthetic-calibration
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit
from scipy.optimize import minimize_scalar

np.random.seed(123)

n_models = 60
n_items = 50

# True model abilities
theta = np.random.normal(0, 1, n_models)

# Human-authored items: broad difficulty range
beta_human = np.random.uniform(-2.5, 2.5, n_items)
alpha_human = np.random.uniform(0.8, 2.0, n_items)

# Synthetic items (well-calibrated): similar difficulty, slight offset
offset = 0.3  # synthetic items slightly easier
beta_syn_good = beta_human + offset + np.random.normal(0, 0.3, n_items)
alpha_syn_good = alpha_human + np.random.normal(0, 0.15, n_items)
```

```
alpha_syn_good = np.clip(alpha_syn_good, 0.3, 3.0)

# Synthetic items (poorly calibrated): mode collapse + circularity
beta_syn_bad = np.random.normal(0, 0.5, n_items)  # narrow difficulty range
alpha_syn_bad = np.random.uniform(0.5, 1.0, n_items)  # low discrimination

# Simulate responses
def simulate_2pl(theta, alpha, beta):
    n_m, n_i = len(theta), len(alpha)
    resp = np.zeros((n_m, n_i))
    for j in range(n_m):
        p = expit(alpha * (theta[j] - beta))
        resp[j] = np.random.binomial(1, p)
    return resp

resp_human = simulate_2pl(theta, alpha_human, beta_human)
resp_syn_good = simulate_2pl(theta, alpha_syn_good, beta_syn_good)
resp_syn_bad = simulate_2pl(theta, alpha_syn_bad, beta_syn_bad)

# Estimate item difficulties (proportion correct as proxy)
p_human = resp_human.mean(axis=0)
p_syn_good = resp_syn_good.mean(axis=0)
p_syn_bad = resp_syn_bad.mean(axis=0)

# Convert to logit scale for comparison
def prop_to_logit(p):
    p = np.clip(p, 0.01, 0.99)
    return -np.log(p / (1 - p))

logit_human = prop_to_logit(p_human)
logit_syn_good = prop_to_logit(p_syn_good)
logit_syn_bad = prop_to_logit(p_syn_bad)

# --- Plot ---
fig, axes = plt.subplots(1, 3, figsize=(6, 2))

# Panel 1: Good calibration transfer
axes[0].scatter(logit_human, logit_syn_good, s=12, alpha=0.6, color='#45BF7C',
 ↪  edgecolor='none')
r_good = np.corrcoef(logit_human, logit_syn_good)[0, 1]
lims = [-4, 4]
axes[0].plot(lims, lims, '--', color='gray', linewidth=0.8)
axes[0].set_xlabel('Human item difficulty')
axes[0].set_ylabel('Synthetic item difficulty')
axes[0].set_title(f'Good transfer (r = {r_good:.2f})')
axes[0].set_xlim(lims)
axes[0].set_ylim(lims)

# Panel 2: Poor calibration transfer
```

```
axes[1].scatter(logit_human, logit_syn_bad, s=12, alpha=0.6, color='#E8637A',
 ↪  edgecolor='none')
r_bad = np.corrcoef(logit_human, logit_syn_bad)[0, 1]
axes[1].plot(lims, lims, '--', color='gray', linewidth=0.8)
axes[1].set_xlabel('Human item difficulty')
axes[1].set_ylabel('Synthetic item difficulty')
axes[1].set_title(f'Poor transfer (r = {r_bad:.2f})')
axes[1].set_xlim(lims)
axes[1].set_ylim(lims)

# Panel 3: Difficulty distributions
bins = np.linspace(-3, 3, 25)
axes[2].hist(beta_human, bins=bins, alpha=0.5, color='#5B8DEE', density=True,
 ↪  label='Human')
axes[2].hist(beta_syn_good, bins=bins, alpha=0.5, color='#45BF7C', density=True,
 ↪  label='Syn (good)')
axes[2].hist(beta_syn_bad, bins=bins, alpha=0.5, color='#E8637A', density=True,
 ↪  label='Syn (bad)')
axes[2].set_xlabel(r'Item difficulty ($\beta$)')
axes[2].set_ylabel('Density')
axes[2].set_title('Difficulty distributions')
axes[2].legend()

plt.tight_layout()
plt.show()

print("Calibration Transfer Analysis:")
print(f"  Good synthetic items: r = {r_good:.2f} with human items")
print(f"    Mean difficulty offset: {logit_syn_good.mean() - logit_human.mean():.2f}")
print(f"  Poor synthetic items: r = {r_bad:.2f} with human items")
print(f"    Difficulty SD - human: {beta_human.std():.2f}, "
      f"good syn: {beta_syn_good.std():.2f}, bad syn: {beta_syn_bad.std():.2f}")
```

The three panels illustrate the calibration transfer problem. **Left**: well-designed synthetic items preserve the rank ordering of difficulty—the correlation with human item difficulties is high and the points cluster near the diagonal (with a small constant offset). **Center**: poorly designed synthetic items (mode-collapsed, narrow difficulty range) lose the rank ordering entirely. **Right**: the difficulty distributions reveal the mechanism: good synthetic items cover the same difficulty range as human items, while poor synthetic items collapse to a narrow band around zero.

The practical implication is that synthetic item quality must be *validated* against human-authored items before the synthetic items can be trusted for evaluation. A small human calibration sample (20–50 items) suffices to check whether calibration transfers. If the correlation between synthetic and human item difficulties is high ($r > 0.8$), the synthetic items can be used with a simple offset correction. If the correlation is low, the synthetic items are measuring something different and should not be substituted for human items.

12.5 Evaluation at Scale

12.5.1 The Hybrid Approach

The tension between synthetic and human evaluation resolves in a hybrid approach: use a *large* synthetic evaluation (high coverage, low cost per item) combined with a *small* human evaluation (gold-standard quality, high cost per item). The statistical challenge is combining the two data sources to produce an estimate that is better than either alone.

Let $\hat{\mu}_{\text{syn}}$ be the mean performance estimated from N synthetic items, and $\hat{\mu}_{\text{human}}$ be the mean performance estimated from n human-evaluated items ($n \ll N$). The synthetic estimate may be biased—due to evaluation circularity, mode collapse, or distribution mismatch—but it provides a useful *proxy* for the true performance.

12.5.2 Prediction-Powered Inference

Prediction-powered inference (PPI), introduced by Angelopoulos et al. (2023) and connected to the off-policy evaluation framework in Section 9.4.3, provides a principled way to combine the two data sources.

The key idea is simple. Let Y_i be the true (human-judged) response on item i and $\hat{Y}_i$ be the synthetic (LLM-judged) response on the same item. We observe $\hat{Y}_i$ for all N items but Y_i for only a subset of n items. The PPI estimator is:

$$\hat{\mu}_{\text{PPI}} = \frac{1}{N}\sum_{i=1}^{N}\hat{Y}_i + \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i) \tag{12.5}$$

The first term is the synthetic estimate over the full dataset. The second term is a **bias correction** computed on the labeled subset: the average discrepancy between human and synthetic judgments. If the synthetic judgments are unbiased ($\mathbb{E}[\hat{Y}_i] = \mathbb{E}[Y_i]$), the correction term averages to zero and $\hat{\mu}_{\text{PPI}} \approx \hat{\mu}_{\text{syn}}$ with the variance of the larger sample. If the synthetic judgments are biased, the correction removes the bias, and the variance of $\hat{\mu}_{\text{PPI}}$ is governed by the variance of the *residual* $Y_i - \hat{Y}_i$, which is small when the synthetic judgments are good proxies.

DEFINITION: PREDICTION-POWERED INFERENCE

Given a large *unlabeled* dataset of size N with predictions $\{\hat{Y}_i\}_{i=1}^{N}$ and a small *labeled* dataset of size n with both predictions $\{\hat{Y}_i\}$ and true labels $\{Y_i\}$:

1. The PPI estimator of the population mean is $\hat{\mu}_{\text{PPI}} = \hat{\mu}_N + (\bar{Y}_n - \hat{\bar{Y}}_n)$, where $\hat{\mu}_N = N^{-1}\sum_{i=1}^{N}\hat{Y}_i$, $\bar{Y}_n = n^{-1}\sum_{i=1}^{n}Y_i$, and $\hat{\bar{Y}}_n = n^{-1}\sum_{i=1}^{n}\hat{Y}_i$.
2. The variance is $\text{Var}(\hat{\mu}_{\text{PPI}}) = \frac{\sigma^2_{\hat{Y}}}{N} + \frac{\sigma^2_{Y-\hat{Y}}}{n}$.
3. The confidence interval is $\hat{\mu}_{\text{PPI}} \pm z_{\alpha/2}\sqrt{\frac{\hat{\sigma}^2_{\hat{Y}}}{N} + \frac{\hat{\sigma}^2_{Y-\hat{Y}}}{n}}$.

When the predictor $\hat{Y}$ is a good proxy for Y, the residual variance $\sigma^2_{Y-\hat{Y}}$ is small, so PPI achieves the confidence interval width of the large sample with the unbiasedness guarantee of the small sample.

12.5.3 Active Evaluation

Not all items benefit equally from human evaluation. If synthetic and human judgments agree on easy items (where the model clearly succeeds or fails), the correction term for those items is near zero. Human evaluation effort is most valuable on *ambiguous* items—those where the synthetic judgment is uncertain or where the model's response is borderline.

This motivates an **active evaluation** strategy: use information-theoretic criteria to select which items receive human judgment. Specifically, prioritize items where:

1. The synthetic judge's confidence is lowest (high entropy of $P(\hat{Y}_i \mid X_i)$).
2. The predicted correction $Y_i - \hat{Y}_i$ has high variance given the item's features.

This connects to the optimal design literature and to the adaptive testing framework from Section 6.2.2: in both cases, we are selecting which measurements to make in order to maximize information per unit cost.

The following simulation demonstrates PPI-style correction: a large synthetic evaluation combined with a small human evaluation corrects bias and narrows confidence intervals.

```
#| label: ppi-correction
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(314)

# --- Setup ---
N_total = 2000   # total items with synthetic judgments
n_human = 80     # items with human judgments
true_accuracy = 0.62  # true model performance

# Synthetic judgments: biased (overestimates by ~5%)
bias = 0.05
noise_syn = 0.15  # noise in synthetic judgments
Y_true = np.random.binomial(1, true_accuracy, N_total).astype(float)
Y_syn = Y_true + bias + np.random.normal(0, noise_syn, N_total)
Y_syn = np.clip(Y_syn, 0, 1)

# Select human-evaluated subset (random)
human_idx = np.random.choice(N_total, n_human, replace=False)
Y_human = Y_true[human_idx]
Y_syn_human = Y_syn[human_idx]
```

```
24
25  # --- Estimators ---
26  # 1. Synthetic only (biased)
27  mu_syn = Y_syn.mean()
28
29  # 2. Human only (unbiased but high variance)
30  mu_human = Y_human.mean()
31
32  # 3. PPI (combines both)
33  mu_ppi = mu_syn + (Y_human - Y_syn_human).mean()
34
35  # --- Confidence intervals ---
36  n_bootstrap = 2000
37  boot_syn = np.zeros(n_bootstrap)
38  boot_human = np.zeros(n_bootstrap)
39  boot_ppi = np.zeros(n_bootstrap)
40
41  for b in range(n_bootstrap):
42      # Bootstrap synthetic
43      idx_s = np.random.choice(N_total, N_total, replace=True)
44      boot_syn[b] = Y_syn[idx_s].mean()
45
46      # Bootstrap human
47      idx_h = np.random.choice(n_human, n_human, replace=True)
48      boot_human[b] = Y_human[idx_h].mean()
49
50      # Bootstrap PPI
51      idx_s2 = np.random.choice(N_total, N_total, replace=True)
52      idx_h2 = np.random.choice(n_human, n_human, replace=True)
53      boot_ppi[b] = Y_syn[idx_s2].mean() + (Y_human[idx_h2] -
    ↪  Y_syn_human[idx_h2]).mean()
54
55  ci_syn = np.percentile(boot_syn, [2.5, 97.5])
56  ci_human = np.percentile(boot_human, [2.5, 97.5])
57  ci_ppi = np.percentile(boot_ppi, [2.5, 97.5])
58
59  # --- Plot ---
60  fig, axes = plt.subplots(1, 2, figsize=(6, 2))
61
62  # Panel 1: Point estimates and CIs
63  methods = ['Synthetic\n(N=2000)', 'Human\n(n=80)', 'PPI\n(N+n)']
64  estimates = [mu_syn, mu_human, mu_ppi]
65  cis_low = [ci_syn[0], ci_human[0], ci_ppi[0]]
66  cis_high = [ci_syn[1], ci_human[1], ci_ppi[1]]
67  colors = ['#E8637A', '#5B8DEE', '#45BF7C']
68
69  for i, (m, est, lo, hi, col) in enumerate(zip(methods, estimates, cis_low, cis_high,
    ↪  colors)):
70      axes[0].errorbar(est, i, xerr=[[est - lo], [hi - est]], fmt='o',
```

```
                      color=col, capsize=4, markersize=5, linewidth=1.5)
axes[0].axvline(true_accuracy, color='gray', linestyle='--', linewidth=1, label=f'True
 ↪  = {true_accuracy}')
axes[0].set_yticks(range(3))
axes[0].set_yticklabels(methods)
axes[0].set_xlabel('Estimated accuracy')
axes[0].set_title('Point estimates & 95% CIs')
axes[0].legend()

# Panel 2: CI widths
widths = [ci_syn[1] - ci_syn[0], ci_human[1] - ci_human[0], ci_ppi[1] - ci_ppi[0]]
bars = axes[1].barh(range(3), widths, color=colors, alpha=0.7, height=0.5)
axes[1].set_yticks(range(3))
axes[1].set_yticklabels(methods)
axes[1].set_xlabel('95% CI width')
axes[1].set_title('Confidence interval comparison')

for i, w in enumerate(widths):
    axes[1].text(w + 0.002, i, f'{w:.3f}', va='center')

plt.tight_layout()
plt.show()

print("Prediction-Powered Inference Results:")
print(f"  True accuracy: {true_accuracy:.3f}")
print(f"  Synthetic only: {mu_syn:.3f}  CI: [{ci_syn[0]:.3f}, {ci_syn[1]:.3f}]  "
      f"width: {ci_syn[1]-ci_syn[0]:.3f}")
print(f"  Human only:     {mu_human:.3f}  CI: [{ci_human[0]:.3f}, {ci_human[1]:.3f}]
 ↪  "
      f"width: {ci_human[1]-ci_human[0]:.3f}")
print(f"  PPI:            {mu_ppi:.3f}  CI: [{ci_ppi[0]:.3f}, {ci_ppi[1]:.3f}]  "
      f"width: {ci_ppi[1]-ci_ppi[0]:.3f}")
print(f"\n  Synthetic bias: {mu_syn - true_accuracy:+.3f}")
print(f"  PPI bias:       {mu_ppi - true_accuracy:+.3f}")
print(f"  CI width reduction (PPI vs human-only): "
      f"{(1 - widths[2]/widths[1])*100:.0f}%")
```

The results demonstrate the three-way comparison. **Left panel**: the synthetic estimator (red) has a narrow confidence interval but is biased upward—it does not cover the true accuracy (dashed line). The human-only estimator (blue) is centered on the true value but has a wide confidence interval due to the small sample. The PPI estimator (green) achieves both: it is approximately unbiased (the correction removes the synthetic bias) and has a confidence interval substantially narrower than the human-only estimate. **Right panel**: the CI widths confirm the efficiency gain—PPI inherits the narrow width from the large synthetic sample while maintaining the unbiasedness guarantee from the human sample.

12.6 Putting It Together: An Adversarial Evaluation Pipeline

The tools developed in this chapter and the preceding ones can be composed into a principled adversarial evaluation pipeline. We describe the pipeline as a sequence of stages, each grounded in the measurement framework.

Stage 1: Build a Red-Team Item Bank. Start with a test specification (Section 12.1) that defines the attack taxonomy and coverage requirements. Generate candidate items through a combination of expert red-teamers (depth) and LLM-based generation (Section 12.4) (breadth). Calibrate item parameters by administering the full bank to a reference panel of models, fitting a multidimensional IRT model (Section 12.3) to estimate item difficulties and discriminations.

Stage 2: Adversarial Adaptive Testing. For each new model, run computerized adaptive testing (Section 6.2.2) using the calibrated item bank. The adaptive algorithm selects items that maximize Fisher information at the model's current estimated robustness level, concentrating effort in the informative region of the attack space. After 30–50 adaptively selected items, produce a robustness estimate $\widehat{\theta}_j^{(\text{adv})}$ with a known standard error.

Stage 3: Synthetic Augmentation. Augment the human-calibrated item bank with LLM-generated items for broader coverage. Validate calibration transfer (Section 12.4.3) by checking that synthetic item difficulties correlate with human item difficulties ($r > 0.8$). Use the synthetic items for coarse screening and the calibrated items for precise measurement.

Stage 4: PPI Correction. For aggregate statistics (e.g., "what fraction of red-team attacks does the model resist?"), use prediction-powered inference (Section 12.5.2) to combine synthetic-judged responses (large N, potentially biased) with human-judged responses (small n, unbiased). This produces confidence intervals that are both narrow and honest.

Stage 5: Conformal Coverage. Apply conformal prediction (Section 9.5) to construct prediction sets for individual items: "with 95% confidence, this model's probability of resisting this attack category is in $[0.3, 0.7]$." Under covariate shift between the calibration and deployment item distributions, use weighted conformal prediction (Section 9.4.4) to maintain coverage guarantees.

This pipeline connects to the causal audit framework from Section 9.6: each stage addresses a potential threat to validity. The item bank ensures content validity. Adaptive testing ensures measurement precision. Synthetic augmentation addresses scalability. PPI correction addresses bias. Conformal prediction addresses uncertainty quantification. Together, they produce adversarial evaluations that are not only thorough but *psychometrically defensible*.

KEY INSIGHT: COMPOSABILITY OF MEASUREMENT TOOLS

The power of the measurement framework developed across this book is its *composability*. Red-teaming (Section 12.1) produces items; IRT (Section 12.1.2) calibrates them; CAT (Section 6.2.2) administers them efficiently; synthetic data (Section 12.4) scales them; PPI (Section 12.5.2) corrects for bias; conformal prediction (Section 9.5) quantifies uncertainty. Each tool solves a specific problem. The pipeline composes them into a system that is greater than the sum of its parts.

12.7 Discussion Questions

1. **Coverage vs. depth in red-teaming**. A company has a fixed budget of 500 person-hours for red-teaming a new model. Should they hire 50 non-expert testers for 10 hours each, or 5 expert red-teamers for 100 hours each? Frame the answer in terms of content validity and item difficulty.

2. **Evaluation circularity**. A team uses GPT-4 to generate evaluation items for GPT-4o. They argue that since GPT-4o is a different model, there is no circularity problem. Under what conditions is this argument valid? When might it fail?

3. **Adversarial robustness as a latent trait**. Is it meaningful to assign a single "adversarial robustness" score to a model, or should robustness always be reported per attack category? Relate the answer to the unidimensionality assumption in IRT.

4. **PPI and evaluator agreement**. Prediction-powered inference assumes that the synthetic and human judgments are measured on the same scale. What happens if the LLM judge uses a different scoring rubric (e.g., binary pass/fail) than the human judges (e.g., 1–5 Likert scale)? How should the PPI framework be modified?

5. **Dynamic item banks**. Red-team item banks become less useful over time as models are trained to resist known attacks. How should an item bank be maintained? Draw an analogy to test security practices in educational testing.

6. **Standardizing ASR comparisons**. A red-teaming leaderboard wants to rank jailbreak methods by effectiveness. Using the framework from Section 12.2, propose a standardized evaluation protocol that ensures conceptual coherence and measurement validity. What should be fixed across submissions (aggregation rule, judge, prompt set)? What can vary?

7. **Distributional vs. aggregate reporting**. Schaeffer et al. (2025) show that the aggregate Top-1 ASR is fully determined by the distribution of per-prompt success probabilities. Should red-teaming studies report the full distribution of per-prompt success rates rather than (or in addition to) a single ASR number? What would this look like in practice, and how does it connect to the IRT perspective where per-prompt success is $\sigma(\theta - \beta_i)$?

12.8 Bibliographic Notes

The use of measurement science in adversarial evaluation is relatively recent. Chouldechova et al. (2026) provided a foundational critique of ASR comparisons, showing that failures of conceptual coherence (different aggregation rules) and measurement validity (differential judge error) undermine most published ASR comparisons. Their work formalizes red-teaming within the social science measurement framework of Adcock and Collier (2001), connecting to the validity concepts developed in Chapter 1. Wallach et al. (2025) further argue that evaluating generative AI systems is fundamentally a social science measurement challenge.

Perez et al. (2022) pioneered the use of LLMs for automated red-teaming, generating adversarial prompts at scale and analyzing the distribution of discovered failures. Ganguli et al. (2022)

provided one of the most comprehensive structured red-teaming efforts, demonstrating scaling behaviors and establishing best practices for human red-team evaluations.

The adversarial filtering approach—using models to select items that fool current systems—was introduced by Zellers et al. (2019) in the context of the HellaSwag benchmark, where adversarial filtering produces items that are trivially easy for humans but challenging for language models. Bartolo et al. (2021) extended this with DynaBench, a platform for dynamic, human-in-the-loop benchmark construction where red-teamers specifically target current model weaknesses.

Ribeiro et al. (2020) proposed a structured testing methodology for NLP models organized around capabilities and test types, providing a practical template for adversarial item banks. Their minimum functionality tests, invariance tests, and directional expectation tests correspond to items with different psychometric properties.

Prediction-powered inference was formalized by Angelopoulos et al. (2023), who showed that combining a large prediction dataset with a small labeled dataset yields valid statistical inference with the confidence interval width governed by the prediction-label residual variance rather than the label variance. The connection to evaluation is natural: synthetic judgments serve as predictions and human judgments as labels.

The multidimensional IRT framework used in Section 12.3 builds on foundational work in psychometric modeling. For a comprehensive treatment, see Embretson and Reise (2000). The application of IRT to AI evaluation is surveyed in Martı□nez-Plumed et al. (2024).

The distributional theory connecting per-problem success rates to aggregate power-law scaling is developed by Schaeffer et al. (2025), who show that $-\log(\text{pass@}k) \sim k^{-\alpha}$ if and only if the distribution of single-attempt success probabilities has a power-law left tail with exponent $\alpha - 1$. This resolves the apparent paradox between exponential per-problem scaling and polynomial aggregate scaling. S. Truong, Tu, et al. (2025) integrate IRT into the scaling law framework via Beta-IRT, which models empirical probability responses (rather than binary) using a Beta loss. Their Item Response Scaling Laws reduce parameter complexity from $O(M \times N)$ to $O(M + N)$ by factorizing model ability from question difficulty, enabling cross-benchmark transfer of ability estimates and achieving comparable scaling predictions with 99.9% fewer evaluation queries.

12.9 Exercises

Exercise 1 (Red-team test specification). Write a test specification for red-teaming a medical chatbot. The specification should include: (a) an attack taxonomy with at least 5 categories, (b) 3 items per category with target difficulty levels (easy, medium, hard), and (c) a scoring rubric. Discuss which categories pose the greatest validity threats if undersampled.

Exercise 2 (Adversarial IRT simulation). Extend the simulation in Section 12.3 to $K = 3$ latent dimensions: standard accuracy, adversarial robustness, and prompt sensitivity. Generate 60 items per dimension. Fit a unidimensional IRT model to all 180 items and compare the ability

estimates to the true three-dimensional traits. Show that the unidimensional model produces misleading rankings when the inter-dimension correlations are low ($\rho < 0.3$).

Exercise 3 (Calibration transfer). In the synthetic calibration simulation (Section 12.4.3), the "good" synthetic items are generated by adding Gaussian noise to the human item parameters. Replace this with a more realistic model: generate synthetic items by sampling from a *narrower* difficulty distribution (mimicking mode collapse) with probability p and from the true distribution with probability $1 - p$. Plot the calibration transfer correlation as a function of p and identify the threshold at which transfer breaks down ($r < 0.8$).

Exercise 4 (PPI with varying bias). Modify the PPI simulation to sweep the synthetic bias from -0.15 to $+0.15$. For each bias level, compute (a) the PPI point estimate, (b) the 95% confidence interval, and (c) the coverage probability (does the CI contain the true value?). Verify that PPI maintains nominal coverage regardless of the bias magnitude, while the synthetic-only estimator's coverage degrades as bias increases.

Exercise 5 (Adaptive adversarial testing). Implement a simple adaptive adversarial testing algorithm. Start with a calibrated item bank of 200 adversarial items with known Rasch difficulties. For each model: (a) initialize $\widehat{\theta}^{(\text{adv})} = 0$, (b) select the item with difficulty closest to $\widehat{\theta}^{(\text{adv})}$, (c) simulate the response, (d) update $\widehat{\theta}^{(\text{adv})}$ using maximum likelihood, (e) repeat for 30 items. Compare the estimation error to a non-adaptive baseline (30 random items) across 100 simulated models.

Exercise 6 (Active evaluation design). In the PPI framework, suppose it is possible to choose *which* n items to send for human evaluation (instead of selecting randomly). Propose an active selection criterion based on the synthetic judge's uncertainty. Implement and compare against random selection: does targeted human evaluation reduce the PPI confidence interval width?

Exercise 7 (ASR aggregation and judge error). Consider a model with true per-prompt success probabilities $p_i \sim \text{Beta}(a, b)$ for $i = 1, \ldots, 200$. (a) Using the result from Schaeffer et al. (2025), show that the expected Top-1 of K ASR satisfies $-\log(\mathbb{E}[\text{ASR}_{\text{Top-1}}(K)]) \sim K^{-a}$ for large K, so the left-tail parameter a controls the power-law exponent. (b) Now suppose the judge has TPR = 0.9 and FPR = ϕ. Derive the expected observed Top-1 ASR as a function of ϕ and K. (c) Simulate the scenario for $a = 0.3, b = 3, K \in \{1, 10, 50, 200\}$, and $\phi \in \{0, 0.05, 0.10, 0.15\}$. Create a heatmap showing how the compound effect of aggregation and judge error inflates the observed ASR. At what (K, ϕ) does the observed ASR exceed 0.9 even though the true one-shot ASR is below 0.1? (d) Fit a Beta distribution to the per-prompt success rates and use the distributional estimator to predict $\text{ASR}_{\text{Top-1}}(K)$ for K up to 10,000. Compare to the empirical estimate: how much compute does the distributional approach save?

Exercise 8 (End-to-end pipeline). Implement the full adversarial evaluation pipeline from Section 12.6 for a simulated scenario. Generate a 500-item adversarial bank with known 2PL parameters. Run adaptive testing on 50 models. Augment with 2000 synthetic items (with 10% mode-collapsed items). Apply PPI correction using 100 human-evaluated items. Report: (a) ability estimates with standard errors, (b) calibration transfer correlation, and (c) the bias reduction from PPI.

13 Conclusion

This chapter synthesizes the book's main themes, identifies open challenges in AI measurement science, describes capstone projects suitable for a one-quarter course, and offers a practitioner's checklist for designing evaluations.

13.1 Our Approach

13.1.1 Foundations from Measurement Science

The central premise of this book is that AI evaluation is a *measurement* problem, not merely an engineering one. When we assign a number to a model — accuracy, Elo rating, pass rate — we are making a claim about a latent construct (ability, safety, reasoning) that we cannot directly observe. The science of such claims has been developed over a century in psychometrics, educational testing, and the philosophy of measurement. We have drawn on this tradition throughout.

Chapter 3 introduced the mathematical models that formalize the relationship between latent ability and observed responses: Item Response Theory (Rasch, 2PL, 3PL), factor models, and paired-comparison systems (Bradley-Terry, Elo). A recurring theme is that these models are not interchangeable — each encodes assumptions about the structure of ability and the nature of items. The Rasch model's sufficiency and specific objectivity properties justify using sum scores as measurements; the 2PL and 3PL models trade these properties for descriptive flexibility. The choice of model is a choice about what we are willing to assume.

13.1.2 From Models to Methods

Chapter 4 showed how to estimate latent parameters from evaluation data via maximum likelihood, EM algorithms, and Bayesian inference. Chapter 6 developed the design principles — Fisher information, computerized adaptive testing, D-optimal design — that make evaluation efficient. Together, these chapters establish that we can measure AI systems with far fewer queries than brute-force evaluation requires: calibrated item banks and adaptive algorithms yield precise ability estimates using a fraction of the items.

13.1.3 Reliability, Validity, and Causality

The transition from Part I to Part II marked a shift from *how* to measure to *how well* we measure. Chapter 5 decomposed the noise in AI evaluation into its constituent sources — sampling stochasticity, prompt sensitivity, annotator variability, item sampling — and provided tools (Generalizability Theory, IRT-based reliability) for quantifying and reducing each. Chapter 1 asked the harder question: even if measurements are precise, do they measure what we intend? Content validity, criterion validity, construct validity, differential item functioning, and benchmark contamination diagnostics all address different facets of this question. Chapter 9 connected these ideas to formal causal reasoning, showing when benchmark results generalize across contexts and how to correct for distribution shift.

13.1.4 Design, Strategy, and Adversarial Evaluation

Part III moved from the science of measurement to its strategic and practical dimensions. Chapter 11 formalized evaluation as a game between evaluators and model builders, showing how strategic behavior (Goodhart's Law, benchmark gaming) distorts measurement and how mechanism design can restore alignment. Chapter 12 applied the full measurement framework to adversarial evaluation, showing that red-teaming is a measurement problem — with all the attendant concerns about validity, reliability, and construct definition.

13.1.5 The Critical Turn

A thread running through every chapter is that technical choices in evaluation are never value-neutral. Which items we include defines the construct (Chapter 1). Which models we test and how we sample them shapes the ability scale (Chapter 3). How we aggregate across annotators determines whose judgments count (Chapter 5). How we design incentives determines what model builders optimize for (Chapter 11). AI measurement science provides tools for making these choices explicit, principled, and auditable — but the choices themselves remain human judgments.

13.2 Lessons from the Field

13.2.1 What Worked

IRT models are surprisingly effective for AI evaluation. Despite being developed for human testing, Rasch and 2PL models fit AI benchmark data well. The key insight from Chapter 3 is that the same mathematical structure — a latent ability interacting with item difficulty through a logistic function — describes both human and AI response patterns. This is not because AI systems are "like humans" but because the statistical structure of evaluation data (binary responses to items of varying difficulty) is the same regardless of the test-taker.

Adaptive testing dramatically reduces evaluation cost. The efficiency gains from CAT (Chapter 6) are not incremental. S. Truong, Tu, et al. (2025) demonstrate that 50 adaptively selected

questions can match the decision accuracy of thousands of uniformly administered questions — a 99.9% reduction in the query budget. This suggests that most of the compute currently spent on AI evaluation is wasted on uninformative items.

Reliability diagnostics catch real problems. Item-level statistics derived from IRT — item-total correlations, tetrachoric correlations, Mokken scalability — are not merely theoretical diagnostics. S. T. Truong et al. (2025) showed that flagging items with anomalous statistics achieves up to 84% precision in detecting genuine benchmark errors: incorrect answer keys, ambiguous wording, and grading bugs.

13.2.2 What Surprised Us

Benchmarks are noisier than they appear. The Generalizability Theory analysis in Chapter 5 reveals that what looks like stable model ranking is often an artifact of large item counts averaging over substantial per-item noise. When decomposed, prompt sensitivity and annotator variability are often larger than the differences between models we claim to distinguish.

Validity is the hard part. Reliability is a solved problem in the sense that we have excellent tools for quantifying and improving it. Validity is not. The question "Does this benchmark measure what it claims to measure?" requires domain expertise, theoretical commitments, and empirical evidence that no purely statistical procedure can provide. The diagnostics in Chapter 1 help, but they are necessary conditions, not sufficient ones.

Scaling laws and measurement theory are deeply connected. The observation by Schaeffer et al. (2025) that per-problem exponential scaling aggregates to power-law scaling — because item difficulties follow a heavy-tailed distribution — is a statement about item response theory in disguise. The distribution of item difficulties determines the aggregate scaling behavior. S. Truong, Tu, et al. (2025) make this connection explicit, showing that IRT ability θ scales linearly with log(FLOP) and that item parameters transfer across benchmarks. Measurement theory does not just evaluate models — it characterizes how they improve.

13.2.3 What Remains Difficult

Construct definition for AI is unresolved. In human testing, constructs like "verbal reasoning" or "mathematical ability" have decades of theoretical development and empirical validation. For AI, we are still debating what "reasoning" means, whether "understanding" is coherent, and how to distinguish genuine capability from surface-level pattern matching. Without clear construct definitions, validity analysis has no target.

Benchmark contamination is an arms race. The diagnostic tools in Chapter 1 can detect some forms of contamination after the fact, but preventing contamination requires ongoing effort: holdout sets, dynamic benchmarks, and adversarial construction. The fundamental tension is that useful benchmarks must be public enough to be widely adopted but private enough to avoid being gamed.

Evaluation of generative output lacks ground truth. Much of this book assumes binary (correct/incorrect) or probability-valued responses. But the most important AI capabilities — open-ended generation, creative problem-solving, multi-turn dialogue — produce outputs where "correctness" is ill-defined. Extending measurement theory to these settings is an open frontier.

13.3 Open Challenges

13.3.1 Beyond Binary Responses

Standard IRT models assume binary or bounded-probability responses. But modern AI evaluation increasingly involves rubric-based scoring (1–5 scales), preference judgments (pairwise comparisons), and continuous quality metrics (BLEU, ROUGE, reward model scores). Extending the measurement framework to these response types — while preserving the desirable properties of IRT (separability, adaptive testing, cross-benchmark transfer) — is an active research area. Beta-IRT (Chapter 4; S. Truong, Tu, et al. (2025)) is one step in this direction, but a general theory of measurement for mixed response types remains elusive.

13.3.2 Multidimensional Ability and Benchmark Portfolios

The unidimensional models emphasized in this book assume a single latent ability. In practice, AI systems have heterogeneous capabilities: a model may excel at coding but struggle with medical reasoning. Multidimensional IRT and factor models (Chapter 3) provide the mathematical framework, but practical questions remain. How many dimensions are needed? How should benchmark portfolios be designed to efficiently measure multiple abilities simultaneously? How should multidimensional ability profiles be communicated to non-expert stakeholders?

13.3.3 Temporal Dynamics and Evaluation Drift

AI evaluation is not static. Models improve, benchmarks saturate, and the relationship between benchmark performance and real-world utility shifts over time. Chapter 9 addressed distribution shift in a cross-sectional setting, but *longitudinal* evaluation — tracking how models and benchmarks co-evolve — requires new tools. Item parameter drift (when calibrated difficulties become stale), concept drift (when the construct itself changes), and benchmark half-life (when a benchmark loses discriminative power) are all practical problems that the field has barely begun to address.

13.3.4 Evaluation of Agentic Systems

The measurement models in this book were developed for systems that produce a single response to a single prompt. Agentic AI systems — those that take multi-step actions in environments, use tools, and interact with humans over extended time horizons — pose fundamentally different measurement challenges. What is the "item" in an agentic evaluation? What is the "response"?

How do we define and measure reliability when the evaluation environment itself is stochastic and path-dependent?

13.3.5 Scalable Oversight and Recursive Evaluation

As AI systems approach or exceed human performance on specific tasks, the evaluation bottleneck shifts from model capability to evaluator capability. If human judges cannot reliably assess the quality of an AI system's outputs, how do we maintain measurement validity? Approaches including AI-assisted evaluation, debate, and recursive reward modeling all attempt to extend the reach of human judgment, but their measurement properties — reliability, validity, potential for systematic bias — are largely uncharacterized.

13.3.6 Fairness in Evaluation

Evaluation systems can be unfair in multiple ways: items may exhibit differential functioning across model families or deployment contexts (Chapter 1), adaptive testing algorithms may underquery certain regions of the ability space, and benchmark selection may systematically favor certain architectures. The fairness of evaluation has received far less attention than the fairness of the systems being evaluated. Developing measurement-theoretic notions of evaluation fairness — analogous to test fairness in educational testing — is an important open direction.

13.4 Capstone Projects

The following projects are designed for the CS321M (AI Measurement Science) course at Stanford. Each integrates concepts from multiple chapters of this book. Projects are suitable for individuals or pairs and should result in a pre-analysis plan (3–4 pages, NeurIPS format) and a final manuscript (up to 8 pages, NeurIPS format) with reproducible code.

Difficulty ratings: ($\star$) single-person, 4 weeks; ($\star\star$) 1–2 persons, 6 weeks; ($\star\star\star$) 1–2 persons, 8+ weeks, potential research contribution.

Project 1: IRT Model Comparison on Real Benchmarks ($\star$)

Fit Rasch, 2PL, and multidimensional IRT models to a large-scale evaluation dataset (e.g., the Open LLM Leaderboard response matrix or HELM data). Compare models using information criteria, cross-validated log-likelihood, and out-of-sample prediction. Test whether the Rasch model's sufficiency property holds empirically by comparing sum-score-based rankings to IRT ability rankings. Investigate whether items that misfit the Rasch model correspond to known benchmark quality issues.

Key concepts: Chapter 3 (IRT models), Chapter 4 (estimation), Chapter 1 (item fit).

Project 2: Adaptive Testing for Efficient AI Evaluation (⋆)

Using a pre-calibrated item bank from an existing benchmark, implement CAT for AI model evaluation. Compare the efficiency (number of items needed to reach a target standard error) and accuracy (rank correlation with full-benchmark scores) of Fisher-information item selection versus random selection. Investigate how performance degrades when calibration data come from a different generation of models (calibration drift).

Key concepts: Chapter 6 (CAT, Fisher information), Chapter 3 (IRT), Chapter 9 (distribution shift).

Project 3: Reliability Audit of an Evaluation Pipeline (⋆)

Select an evaluation pipeline that involves human or LLM judges (e.g., MT-Bench, AlpacaEval, or Chatbot Arena). Design and conduct a G-study decomposing variance into model, item, judge, and interaction components. Run a D-study to determine the optimal allocation of judges and items under a fixed budget. Report the generalizability coefficient and compare it to the reliability implicitly assumed by published leaderboards.

Key concepts: Chapter 5 (G-theory, D-studies, LLM-as-judge reliability).

Project 4: Benchmark Bug Detection at Scale (⋆⋆)

Apply the diagnostic framework of S. T. Truong et al. (2025) to a benchmark not studied in their paper. Compute item-total correlations, tetrachoric correlations, and Mokken scalability coefficients across a diverse set of LLMs. Flag the top-50 most suspicious items and conduct a manual review. Report precision, analyze the types of errors found, and propose corrections. Investigate how the number and diversity of LLMs affects detection power.

Key concepts: Chapter 3 (sufficiency, Rasch), Chapter 5 (item-total correlation), Chapter 1 (content validity).

Project 5: Validity Analysis of a Domain-Specific Benchmark (⋆⋆)

Choose a domain-specific benchmark (medical, legal, coding, mathematical reasoning). Conduct a comprehensive validity analysis: (a) content validity — does the item pool representatively sample the stated construct? (b) construct validity — does dimensionality analysis support a unidimensional interpretation? (c) criterion validity — do scores correlate with external measures of the construct? (d) DIF analysis — do items function differently across model families?

Key concepts: Chapter 1 (all validity types, DIF), Chapter 3 (factor models, dimensionality).

Project 6: Scaling Laws through the IRT Lens (⋆⋆)

Replicate and extend the Item Response Scaling Laws framework of S. Truong, Tu, et al. (2025). Using publicly available checkpoint evaluation data, fit Beta-IRT models and estimate the relationship between θ and pre-training compute. Test cross-benchmark transfer: estimate θ on one benchmark and predict performance on another. Compare the efficiency of IRT-based scaling estimation to traditional per-benchmark curve fitting.

Key concepts: Chapter 3 (IRT), Chapter 4 (Beta-IRT), Chapter 6 (adaptive testing), Chapter 9 (transfer).

Project 7: Red-Teaming as Measurement (⋆⋆)

Design and calibrate an adversarial item bank for a specific safety domain (e.g., medical misinformation, code injection, social engineering). Fit a multidimensional IRT model with standard and adversarial ability dimensions. Implement adaptive adversarial testing and compare its efficiency to uniform item selection. Analyze the validity of the adversarial construct: does adversarial robustness form a coherent dimension, or does it fragment into domain-specific factors?

Key concepts: Chapter 12 (adversarial IRT, item banks), Chapter 6 (CAT), Chapter 1 (construct validity).

Project 8: ASR Comparisons Under Aggregation and Judge Error (⋆⋆)

Conduct an empirical study of the aggregation and judge-error biases documented by Chouldechova et al. (2026). Using a red-teaming dataset with multiple samples per prompt, estimate: (a) the distribution of per-prompt success probabilities, (b) the power-law exponent under Top-1 aggregation (Schaeffer et al. (2025)), (c) differential judge error rates across target models. Quantify how much of the published variation in ASR across models is attributable to estimand differences versus genuine safety differences.

Key concepts: Chapter 12 (ASR as estimand, judge validity), Chapter 5 (measurement error).

Project 9: Prediction-Powered Evaluation (⋆⋆)

Implement and evaluate the prediction-powered inference (PPI) framework for AI evaluation. Use a large LLM judge as the prediction source and a smaller human-labeled set as ground truth. Compare PPI estimates to synthetic-only and human-only baselines across multiple benchmarks. Investigate: how does PPI performance depend on the quality of the LLM judge? What is the minimum human annotation budget needed for PPI to outperform both baselines?

Key concepts: Chapter 12 (PPI, synthetic data), Chapter 5 (judge reliability), Chapter 9 (doubly robust estimation).

Project 10: Evaluation Design for Agentic Systems (⋆ ⋆ ⋆)

Propose and evaluate a measurement framework for agentic AI systems (e.g., coding agents, web-browsing agents, tool-using assistants). Key challenges include: defining the unit of measurement (what is an "item"?), handling variable-length interactions, and decomposing performance into sub-capabilities. Implement a prototype evaluation using a real agent benchmark and analyze its reliability (via G-theory over tasks, seeds, and environments) and construct validity (via factor analysis of sub-task scores).

Key concepts: Chapter 3 (factor models), Chapter 5 (G-theory), Chapter 1 (construct validity), Chapter 6 (design).

13.5 A Practitioner's Checklist

The following checklist distills the book's recommendations into actionable steps for designing and auditing AI evaluations. It is organized by the evaluation lifecycle.

13.5.1 Construct Definition

- ☐ Write an explicit construct definition: what is the benchmark intended to measure, and what is it *not* intended to measure?
- ☐ Specify the content domain and develop a test blueprint mapping sub-domains to item counts (Chapter 1).
- ☐ Distinguish between the construct (latent ability) and the indicator (observed score). A benchmark score is *evidence about* ability, not ability itself (Chapter 3).

13.5.2 Item Development and Calibration

- ☐ Calibrate items using IRT: estimate difficulty, discrimination, and (if applicable) guessing parameters on a diverse set of models (Chapter 4).
- ☐ Screen for problematic items using item-total correlations, tetrachoric correlations, and fit statistics. Remove or revise items that misfit (Chapter 5; S. T. Truong et al. (2025)).
- ☐ Check for DIF: do items function differently across model families, sizes, or training paradigms? Differential functioning may indicate construct-irrelevant variance (Chapter 1).
- ☐ Audit item content for errors: incorrect keys, ambiguous wording, grading bugs, culturally-embedded assumptions.

13.5.3 Evaluation Design

- ☐ Choose the response format (binary, probability, preference, rubric) that best captures the construct. Binary responses are simplest but discard information; empirical probabilities and rubric scores preserve richer signals (Chapter 4; S. Truong, Tu, et al. (2025)).

- ☐ Consider adaptive testing: if the item bank is calibrated, CAT can reduce the query budget by 90%+ with minimal loss in precision (Chapter 6).
- ☐ For judge-based evaluation, design the rating protocol with reliability in mind: standardize rubrics, randomize presentation order, and plan for multiple judges (Chapter 5).
- ☐ Run a D-study to allocate the evaluation budget optimally across items, judges, and replications (Chapter 5).

13.5.4 Analysis and Reporting

- ☐ Report confidence intervals, not just point estimates. The SEM provides a natural uncertainty band; models whose scores overlap within $\pm 2 \times$ SEM may not be meaningfully different (Chapter 5).
- ☐ Report the reliability of the evaluation (Cronbach's α, G-coefficient, or IRT-based conditional reliability). If reliability is low, ranking differences are uninterpretable.
- ☐ For red-teaming, always report the aggregation rule (one-shot, Top-1 of K) alongside the ASR. Comparisons across aggregation rules are not comparisons of safety (Chapter 12).
- ☐ For judge-based evaluation, report inter-rater agreement disaggregated by model and item type. Differential judge error invalidates comparisons (Chapter 12).

13.5.5 Validity and Generalization

- ☐ Assess content validity: does the item pool representatively sample the intended construct domain?
- ☐ Assess construct validity: does dimensionality analysis support the intended interpretation? Do scores correlate with external criteria?
- ☐ Consider distribution shift: will the evaluation generalize to the deployment context? If not, apply importance weighting or doubly robust corrections (Chapter 9).
- ☐ Document known validity threats and limitations. No evaluation is perfectly valid; transparency about limitations is itself a form of quality assurance.

13.5.6 Maintenance and Monitoring

- ☐ Monitor for benchmark saturation: when the top models all score above 95%, the benchmark no longer discriminates and should be retired or extended.
- ☐ Monitor for contamination: periodically check whether new models have been trained on benchmark data, using the diagnostics from Chapter 1.
- ☐ Recalibrate item parameters periodically: as the model population changes, item difficulties may drift.
- ☐ Maintain a living item bank: retire compromised items, add new items, and recalibrate regularly.

13.6 Scope and Limitations

This book has focused on the statistical and psychometric foundations of AI evaluation. Several important topics are largely out of scope:

Cognitive science of AI systems. We treat AI models as black boxes that produce responses to items. Understanding *why* a model succeeds or fails on a particular item — through mechanistic interpretability, probing, or causal analysis of internal representations — is a complementary but distinct line of inquiry.

Large-scale systems engineering. Evaluation at the scale of modern AI development involves distributed computing, data pipelines, versioning, and infrastructure that we do not cover. The measurement principles in this book apply regardless of scale, but their implementation at scale introduces engineering challenges.

Legal and regulatory frameworks. AI evaluation increasingly intersects with regulation (EU AI Act, NIST AI RMF). We do not cover the legal dimensions of evaluation, though the validity and reliability frameworks developed here provide the scientific basis for regulatory requirements.

Domain-specific evaluation. While we discuss domain-specific examples throughout, we do not provide deep coverage of evaluation in any single domain (healthcare, education, law, finance). Each domain brings its own construct definitions, validity concerns, and stakeholder requirements that warrant dedicated treatment.

Multi-modal evaluation. The models in this book are developed primarily for text-based evaluation. Extending measurement theory to vision, audio, multi-modal, and embodied AI evaluation raises new questions about construct definition and response modeling.

13.7 Final Thought

The gap between what we claim about AI systems and what we can rigorously demonstrate is wide. Benchmarks proliferate, but the science behind them lags. We have the tools — developed over a century of measurement science — to close this gap. Item response theory, factor analysis, reliability theory, validity analysis, causal reasoning, and mechanism design are not merely theoretical curiosities; they are the foundation for making AI evaluation trustworthy.

The challenge is not technical but cultural. The AI community has optimized for speed — new benchmarks, new models, new leaderboards — at the expense of rigor. Adopting measurement science requires slowing down: defining constructs before collecting data, calibrating items before reporting scores, quantifying uncertainty before claiming progress. This is the discipline that separates measurement from mere scoring.

We invite readers to contribute — to develop new measurement methods, to apply existing ones rigorously, to critique evaluations that fall short, and to insist that claims about AI systems rest on solid scientific ground. The stakes are high: the decisions we make about AI deployment,

safety, and regulation depend on the quality of our measurements. Let us make them worthy of the task.

Notation

This appendix collects the notation used throughout the book. Symbols are grouped thematically; the **Introduced** column indicates the chapter where each symbol first appears.

General

Symbol	Meaning	Domain	Introduced
N	Number of persons / models	$\mathbb{N}$	Chapter 3
M	Number of items / questions	$\mathbb{N}$	Chapter 3
Y_{ij}	Binary response of model i to item j (0 = incorrect, 1 = correct)	$\{0, 1\}$	Chapter 3
$S_i = \sum_j Y_{ij}$	Sum score (total correct) for model i	$\{0, 1, \dots, M\}$	Chapter 3
$\sigma(x) = \frac{1}{1+e^{-x}}$	Logistic sigmoid function	$(0, 1)$	Chapter 3
$\Phi(x)$	Standard normal CDF; also a generic monotone link in Section 3.2	$(0, 1)$	Chapter 3

Item Response Theory

Symbol	Meaning	Domain	Introduced
θ_i or U_i	Latent ability of model i	$\mathbb{R}$	Chapter 3
β_j or V_j	Difficulty of item j	$\mathbb{R}$	Chapter 3

Symbol	Meaning	Domain	Introduced
β_j	Item-parameter vector $(\beta_{j1}, \beta_{j2}, \beta_{j3})$ = (difficulty, discrimination, guessing); $\beta_{j1} = \beta_j$	$\mathbb{R}^n$	Chapter 3
$p_{ij}, \hat{p}_{ij}$	Response probability $P(Y_{ij} = 1)$ and its empirical estimate	$[0, 1]$	Chapter 3
f, g	Additive conjoint component functions (ability, difficulty factors)	$\mathbb{R}$	Chapter 3
β_{j2}	Discrimination parameter of item j (component of β_j; conventionally a_j)	$\mathbb{R}^+$	Chapter 3
β_{j3}	Guessing (pseudo-chance) parameter of item j (component of β_j; conventionally c_j)	$[0, 1]$	Chapter 3
d_j	Discrimination parameter (alternative notation, as in 1PL/2PL)	$\mathbb{R}^+$	Chapter 3
z_j	Difficulty parameter (alternative notation)	$\mathbb{R}$	Chapter 3
$I_j(\theta)$	Fisher information for item j at ability θ	$\mathbb{R}^+$	Chapter 6
$\mathcal{J}(\theta)$	Fisher information matrix	$\mathbb{R}^{K \times K}$	Chapter 6

Learning and Estimation

Symbol	Meaning	Domain	Introduced
$\ell(\theta, \beta)$	Log-likelihood function	$\mathbb{R}$	Chapter 4
$\nabla_\theta \ell$	Gradient of log-likelihood w.r.t. ability parameters	$\mathbb{R}^N$	Chapter 4

Symbol	Meaning	Domain	Introduced
$\pi(\theta)$	Prior distribution over abilities	–	Chapter 4
$\pi(\beta)$	Prior distribution over difficulties	–	Chapter 4
$\hat{\theta}_{\text{MLE}}$	Maximum likelihood estimate of ability	$\mathbb{R}^N$	Chapter 4
$\hat{\theta}_{\text{MAP}}$	Maximum a posteriori estimate of ability	$\mathbb{R}^N$	Chapter 4
η	Learning rate	$\mathbb{R}^+$	Chapter 4

Reliability

Symbol	Meaning	Domain	Introduced
X_{ij}	Observed score for model i on occasion / item j	$\mathbb{R}$	Chapter 5
T_i	True score for model i (CTT)	$\mathbb{R}$	Chapter 5
E_{ij}	Error component (CTT)	$\mathbb{R}$	Chapter 5
$\rho_{XX'}$	Reliability coefficient	$[0, 1]$	Chapter 5
α	Cronbach's alpha	$(-\infty, 1]$	Chapter 5
$\sigma_p^2, \sigma_i^2, \sigma_r^2$	Variance components: person (model), item, rater	$\mathbb{R}^+$	Chapter 5
G	Generalizability coefficient	$[0, 1]$	Chapter 5
n_r, n_i	Number of raters, items in a D-study design	$\mathbb{N}$	Chapter 5
κ	Cohen's kappa (inter-rater agreement)	$[-1, 1]$	Chapter 5

Validity

Symbol	Meaning	Domain	Introduced
$g \in \{0, 1\}$	Group membership indicator (DIF analysis)	$\{0, 1\}$	Chapter 1
α_{MH}	Mantel-Haenszel odds ratio	$\mathbb{R}^+$	Chapter 1
λ_k	k-th eigenvalue of the correlation matrix	$\mathbb{R}$	Chapter 1
MNSQ_i	Mean-square fit statistic for item i	$\mathbb{R}^+$	Chapter 1
r_{ij}	Correlation between trait i measured by method j (MTMM)	$[-1, 1]$	Chapter 1

Causality and Distribution Shift

Symbol	Meaning	Domain	Introduced
$\text{do}(X = x)$	Intervention setting variable X to value x	–	Chapter 9
$P^{(s)}, P^{(t)}$	Source (benchmark) / target (deployment) distribution	–	Chapter 9
$\pi_0(a \mid x)$	Logging (benchmark) policy	$[0, 1]$	Chapter 9
$\pi(a \mid x)$	Target (deployment) policy	$[0, 1]$	Chapter 9
$w(x)$	Importance weight $P^{(t)}(x)/P^{(s)}(x)$	$\mathbb{R}^+$	Chapter 9
$\hat{r}(x, a)$	Reward model (e.g., IRT prediction)	$[0, 1]$	Chapter 9
$\hat{V}_{\text{DR}}$	Doubly robust value estimator	$\mathbb{R}$	Chapter 9
$C_\alpha(x)$	Conformal prediction set at level α	subset of $\mathcal{Y}$	Chapter 9

Information and Mechanism Design

Symbol	Meaning	Domain	Introduced
$\mathcal{C}$	Construct: the latent capability being measured	—	Section 11.2
D	Operationalizing distribution (formally π_E)	distribution on F	Section 11.2
D', D''	Public proxy distributions for $\mathcal{C}$	distributions on F	Section 11.6
S	Realized evaluation set, $S \sim D$	$S \subseteq F$	Section 11.2
μ	Social-relevance distribution (construct fully operationalized; Uniform(F) as a special case)	distribution on F	Section 11.2
F	Universe of tasks	finite set, $\|F\| = N$	Section 11.3
F_E, F_M	Evaluator's / builder's task sets	$F_E, F_M \subseteq F$	Section 11.3
π_E, π_M	Evaluator's / builder's sampling distributions	distributions on F	Section 11.3
$f(\theta)$	Task performance function	$[0, 1]$	Section 11.3
$u_E(\theta)$	Evaluator's utility: $\mathbb{E}_{f\sim\mu}[f(\theta)]$	$\mathbb{R}^+$	Section 11.3
k, k^*	Number of (optimal) sampled evaluation tasks per round	$\mathbb{N}$	Section 11.3
ρ, σ_c	Distribution correction rate and bandwidth	$(0, 1] \times \mathbb{R}^+$	Section 11.3
Δ_t	Residual misalignment: $\max_\theta u_E(\theta) - u_E(\theta_t)$	$\mathbb{R}^+$	Section 11.3
$\tilde{\pi}_E$	Builder's belief about the evaluator's sampling	distribution on F	Section 11.3
C	Agent cost (principal-agent model)	$\mathbb{R}^+$	Section 11.3

Symbol	Meaning	Domain	Introduced
b	Principal's value from agent effort	$\mathbb{R}^+$	Section 11.3

Red-Teaming and Adversarial Evaluation

Symbol	Meaning	Domain	Introduced
$\theta_j^{(\text{adv})}$	Adversarial robustness of model j	$\mathbb{R}$	Chapter 12
$\theta_j^{(\text{std})}$	Standard accuracy ability of model j	$\mathbb{R}$	Chapter 12
$\beta_i^{(\text{atk})}$	Attack strength (difficulty) of adversarial item i	$\mathbb{R}$	Chapter 12
δ	Perturbation applied to an input	$\mathcal{B}_\epsilon$	Chapter 12
$\alpha_{s,\mathcal{D}}$	Attack success probability under criterion s and goal distribution $\mathcal{D}$	$[0,1]$	Chapter 12
J	Operational judge of attack success	$\{0,1\}$	Chapter 12
s	Oracle (true) success criterion	$\{0,1\}$	Chapter 12
K	Number of repeated samples in Top-1 aggregation	$\mathbb{N}$	Chapter 12
$\hat{\mu}_{\text{syn}}$	Mean performance estimated from synthetic items	$[0,1]$	Chapter 12
$\hat{\mu}_{\text{human}}$	Mean performance estimated from human-authored items	$[0,1]$	Chapter 12
$\hat{\mu}_{\text{PPI}}$	Prediction-powered inference estimator	$[0,1]$	Chapter 12
n	Number of human-evaluated items	$\mathbb{N}$	Chapter 12

References

Adcock, Robert, and David Collier. 2001. "Measurement Validity: A Shared Standard for Qualitative and Quantitative Research." *American Political Science Review* 95 (3): 529–46.

Aidar Myrzakhan, Zhiqiang Shen, Sondos Mahmoud Bsharat. 2024. "Open-LLM-Leaderboard: From Multi-Choice to Open-Style Questions for LLMs Evaluation, Benchmark, and Arena." *arXiv Preprint arXiv:2406.07545*.

Akerlof, George A. 1970. "The Market for 'Lemons': Quality Uncertainty and the Market Mechanism." *The Quarterly Journal of Economics* 84 (3): 488–500.

Alonso, Iñigo, Maite Oronoz, and Rodrigo Agerri. 2024. "MedExpQA: Multilingual Benchmarking of Large Language Models for Medical Question Answering." *Artificial Intelligence in Medicine* 155: 102938. https://doi.org/10.1016/j.artmed.2024.102938.

Andersen, Erling B. 1970. "Asymptotic Properties of Conditional Maximum-Likelihood Estimators." *Journal of the Royal Statistical Society. Series B (Methodological)* 32: 283–301. https://doi.org/10.1111/j.2517-6161.1970.tb00842.x.

Andersen, Erling B. 1973. "Conditional Inference for Multiple-Choice Questionnaires." *British Journal of Mathematical and Statistical Psychology* 26: 31–44.

Angelopoulos, Anastasios N., Stephen Bates, Emmanuel J. Candès, Michael I. Jordan, and Lihua Lei. 2023. "Prediction-Powered Inference." *Science* 382 (6671): 669–74.

Anthropic. 2025. *Alignment Faking RL Dataset*. Hugging Face dataset. https://huggingface.co/datasets/Anthropic/alignment-faking-rl.

Askell, Amanda, Yuntao Bai, Anna Chen, et al. 2021. *A General Language Assistant as a Laboratory for Alignment*. https://arxiv.org/abs/2112.00861.

Atkinson, Anthony C., Alexander N. Donev, and Randall D. Tobias. 2007. *Optimum Experimental Designs, with SAS*. Oxford University Press.

Aziz, Haris, Markus Brill, Vincent Conitzer, Edith Elkind, Rupert Freeman, and Toby Walsh. 2017. "Justified Representation in Approval-Based Committee Voting." *Social Choice and Welfare* 48: 461–85.

Bai, Yuntao, Andy Jones, Kamal Ndousse, et al. 2022. *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. https://arxiv.org/abs/2204.05862.

Baker, Frank B., and Seock-Ho Kim. 2004. *Item Response Theory: Parameter Estimation Techniques*. 2nd ed. Marcel Dekker.

Barber, Rina Foygel, Emmanuel J. Candès, Aaditya Ramdas, and Ryan J. Tibshirani. 2023. "Conformal Prediction Beyond Exchangeability." *Annals of Statistics* 51 (2): 816–45.

Bareinboim, Elias, and Judea Pearl. 2016. "Causal Inference and the Data-Fusion Problem." *Proceedings of the National Academy of Sciences* 113 (27): 7345–52.

Bartolo, Max, Alastair Roberts, Johannes Welbl, Sebastian Riedel, and Pontus Stenetorp. 2021. "Beat the AI: Investigating Adversarial Human Annotation for Reading Comprehension." *Transactions of the Association for Computational Linguistics* 8: 662–78.

Bastani, Hamsa, Osbert Bastani, Alp Sungu, Haosen Ge, Özge Kabakcı, and Rei Mariman. 2025. "Generative AI Without Guardrails Can Harm Learning." *Proceedings of the National Academy of Sciences*. https://www.pnas.org/doi/10.1073/pnas.2422633122.

Becker, Joel, Nate Rush, Elizabeth Barnes, and David Rein. 2025. *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity*. https://arxiv.org/abs/2507.09089.

Bergemann, Dirk, and Stephen Morris. 2019. "Information Design: A Unified Perspective." *Journal of Economic Literature* 57 (1): 44–95.

Biderman, Stella et al. 2024. "Lessons from the Trenches on Reproducible Evaluation of Language Models." *arXiv Preprint arXiv:2405.14782*.

Birnbaum, Allan. 1968. "Some Latent Trait Models and Their Use in Inferring an Examinee's Ability." In *Statistical Theories of Mental Test Scores*, edited by Frederic M. Lord and Melvin R. Novick. Addison-Wesley.

Blum, Avrim, and Moritz Hardt. 2015. "The Ladder: A Reliable Leaderboard for Machine Learning Competitions." *Proceedings of the 32nd International Conference on Machine Learning*, 1006–14.

Bock, R. Darrell, and Murray Aitkin. 1981. "Marginal Maximum Likelihood Estimation of Item Parameters: Application of an EM Algorithm." *Psychometrika* 46 (4): 443–59. https://doi.org/10.1007/BF02293801.

Bock, R. Darrell, and Milton Lieberman. 1970. "Fitting a Response Model for n Dichotomously Scored Items." *Psychometrika* 35 (2): 179–97. https://doi.org/10.1007/BF02291262.

Bommasani, Rishi, Percy Liang, and Tony Lee. 2024. "Holistic Evaluation of Language Models." *Annals of the New York Academy of Sciences*.

Borkulo, Claudia D. van, Denny Borsboom, Sacha Epskamp, et al. 2014. "A New Method for Constructing Networks from Binary Data." *Scientific Reports* 4: 5918.

Borsboom, Denny. 2005. *Measuring the Mind: Conceptual Issues in Contemporary Psychometrics*. Cambridge University Press.

Bradley, Ralph Allan, and Milton E Terry. 1952a. "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." *Biometrika* 39 (3/4): 324–45.

Bradley, Ralph Allan, and Milton E. Terry. 1952b. "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." *Biometrika* 39 (3/4): 324–45. http://www.jstor.org/stable/2334029.

Braverman, Mark, and Sumegha Garg. 2020. "The Role of Randomness and Noise in Strategic Classification." *Foundations of Responsible Computing (FORC)*.

Brennan, Robert L. 2001. *Generalizability Theory*. Springer.

Brogden, Hubert E. 1977. "The Rasch Model, the Law of Comparative Judgment and Additive Conjoint Measurement." *Psychometrika* 42 (4): 631–34.

Brown, William. 1910. "Some Experimental Results in the Correlation of Mental Abilities." *British Journal of Psychology* 3 (3): 296–322.

California Department of Motor Vehicles. 2024. *Autonomous Vehicle Disengagement Reports*. California DMV regulatory database. https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-disengagement-reports/.

Campbell, Donald T., and Donald W. Fiske. 1959. "Convergent and Discriminant Validation by the Multitrait-Multimethod Matrix." *Psychological Bulletin* 56 (2): 81–105.

Carpenter, Bob, Andrew Gelman, Matthew D. Hoffman, et al. 2017. "Stan: A Probabilistic Programming Language." *Journal of Statistical Software* 76 (1): 1–32. https://doi.org/10.18637/jss.v076.i01.

Chang, Hua-Hua, and Zhiliang Ying. 2009. "Nonlinear Sequential Designs for Logistic Item Response Theory Models with Applications to Computerized Adaptive Tests." *Annals of Statistics* 37 (3): 1466–88.

Chen, Lingjiao, Matei Zaharia, and James Zou. 2024. "How Is ChatGPT's Behavior Changing over Time?" *Harvard Data Science Review* 6 (2). https://doi.org/10.1162/99608f92.5317da47.

Chen, Zhiyu, Wenhu Chen, Charese Smiley, et al. 2022. *FinQA: A Dataset of Numerical Reasoning over Financial Data*. https://arxiv.org/abs/2109.00122.

Chen, Zhiyu, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. 2022. *ConvFinQA: Exploring the Chain of Numerical Reasoning in Conversational Finance Question Answering*. https://arxiv.org/abs/2210.03849.

Chi, Wayne, Valerie Chen, Ryan Shar, et al. 2026. "EditBench: Evaluating LLM Abilities to Perform Real-World Instructed Code Edits." *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=FtL9eEmU6v.

Chiang, Wei-Lin, Lianmin Zheng, Ying Sheng, et al. 2024. *Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference*. https://arxiv.org/abs/2403.04132.

Chouldechova, Alexandra, A. Feder Cooper, Solon Barocas, Abhinav Palia, Dan Vann, and Hanna Wallach. 2026. "Comparison Requires Valid Measurement: Rethinking Attack Success Rate Comparisons in AI Red Teaming." *arXiv Preprint arXiv:2502.00000*.

Cohen, Jacob. 1960. "A Coefficient of Agreement for Nominal Scales." *Educational and Psychological Measurement* 20 (1): 37–46.

Cohen, Jacob. 1968. "Weighted Kappa: Nominal Scale Agreement Provision for Scaled Disagreement or Partial Credit." *Psychological Bulletin* 70 (4): 213–20.

Colombo, Pierre, Benjamin Clavié, Nathan Nogué, and Pablo Piantanida. 2022. "What Are the Best Systems? New Perspectives on NLP Benchmarking." *arXiv Preprint arXiv:2202.03799*.

Cronbach, Lee J. 1951. "Coefficient Alpha and the Internal Structure of Tests." *Psychometrika* 16 (3): 297–334.

Cronbach, Lee J., Goldine C. Gleser, Harinder Nanda, and Nageswari Rajaratnam. 1972. *The Dependability of Behavioral Measurements*. Wiley.

Cronbach, Lee J., and Paul E. Meehl. 1955. "Construct Validity in Psychological Tests." *Psychological Bulletin* 52 (4): 281–302.

Cui, Ganqu, Lifan Yuan, Ning Ding, et al. 2024. *UltraFeedback: Boosting Language Models with Scaled AI Feedback*. https://arxiv.org/abs/2310.01377.

Cui, Kevin, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. 2026. "The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers." *Management Science*. https://pubsonline.informs.org/doi/10.1287/mnsc.2025.00535.

Debenedetti, Edoardo, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. *AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents*. https://arxiv.org/abs/2406.13352.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. "Maximum Likelihood from Incomplete Data via the EM Algorithm." *Journal of the Royal Statistical Society. Series B (Methodological)* 39 (1): 1–38. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x.

Dorn, Diego, Alexandre Variengien, Charbel-Raphaël Segerie, and Vincent Corruble. 2024. "BELLS: A Framework Towards Future Proof Benchmarks for the Evaluation of LLM Safeguards." *arXiv Preprint arXiv:2406.01364*. https://arxiv.org/abs/2406.01364.

Dranove, David, Daniel Kessler, Mark McClellan, and Mark Satterthwaite. 2003. "Is More Information Better? The Effects of 'Report Cards' on Health Care Providers." *Journal of Political Economy* 111 (3): 555–88.

Dudı□k, Miroslav, John Langford, and Lihong Li. 2011. "Doubly Robust Policy Evaluation and Learning." *Proceedings of the 28th International Conference on Machine Learning*, 1097–104.

Dwork, Cynthia, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. 2015. "Generalization in Adaptive Data Analysis and Holdout Reuse." *Advances in Neural Information Processing Systems* 28.

Dwork, Cynthia, and Aaron Roth. 2014. *The Algorithmic Foundations of Differential Privacy*. In *Foundations and Trends in Theoretical Computer Science*, vol. 9. Now Publishers.

Efron, Bradley, and Charles Stein. 1981. "The Jackknife Estimate of Variance." *The Annals of Statistics* 9 (3): 586–96.

Embretson, Susan E., and Steven P. Reise. 2000. *Item Response Theory for Psychologists*. Lawrence Erlbaum Associates.

Fei, Zhiwei, Xiaoyu Shen, Dawei Zhu, et al. 2023. *LawBench: Benchmarking Legal Knowledge of Large Language Models*. https://arxiv.org/abs/2309.16289.

Fenogenova, Alena, Artem Chervyakov, Nikita Martynov, et al. 2024. "MERA: A Comprehensive LLM Evaluation in Russian." *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Bangkok, Thailand), 9920–48. https://doi.org/10.18653/v1/2024.acl-long.534.

Fischer, Gerhard H., and Ivo W. Molenaar. 1995. *Rasch Models: Foundations, Recent Developments, and Applications*. Springer.

Fleiss, Joseph L. 1971. "Measuring Nominal Scale Agreement Among Many Raters." *Psychological Bulletin* 76 (5): 378–82.

Fox, Jean-Paul. 2010. *Bayesian Item Response Modeling: Theory and Applications*. Springer.

Frank, Michael C., Mika Braginsky, Julie Cachia, et al. 2025. *Experimentology: An Open Science Approach to Experimental Psychology Methods*. MIT Press.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2008. "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics* 9 (3): 432–41.

Fu, Chaoyou, Peixian Chen, Yunhang Shen, et al. 2023. *MME: A Comprehensive Evaluation Benchmark for Multimodal Large Language Models*. https://arxiv.org/abs/2306.13394.

Fu, Chaoyou, Yuhan Dai, Yongdong Luo, et al. 2025. "Video-MME: The First-Ever Comprehensive Evaluation Benchmark of Multi-Modal LLMs in Video Analysis." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 24108–18. https://arxiv.org/abs/2405.21075.

Future of Life Institute. 2025. *AI Safety Index: Winter 2025*. Report. https://futureoflife.org/ai-safety-index-winter-2025/.

Ganguli, Deep, Liane Lovitt, Jackson Kernion, et al. 2022. *Red Teaming Language Models to Reduce Harms: Methods, Scaling Behaviors, and Lessons Learned*. https://arxiv.org/abs/2209.07858.

Gao, Leo, John Schulman, and Jacob Hilton. 2023. "Scaling Laws for Reward Model Overoptimization." *Proceedings of the 40th International Conference on Machine Learning*, 10835–66.

Ghosh, Shaona, Prasoon Varshney, Makesh Narsimhan Sreedhar, et al. 2025. "AEGIS2.0: A Diverse AI Safety Dataset and Risks Taxonomy for Alignment of LLM Guardrails." *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)* (Albuquerque, New Mexico), 5992–6026. https://aclanthology.org/2025.naacl-long.306.

González, José Ángel, Ian Borrego Obrador, Álvaro Romo Herrero, et al. 2025. *IberBench: LLM Evaluation on Iberian Languages*. https://arxiv.org/abs/2504.16921.

Goodhart, Charles A. E. 1984. "Problems of Monetary Management: The U.K. Experience." *Monetary Theory and Practice*, 91–121.

Gu, Alex, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. 2024. *CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution*. https://arxiv.org/abs/2401.03065.

Guha, Neel, Julian Nyarko, Daniel E. Ho, et al. 2023. *LegalBench: A Collaboratively Built Benchmark for Measuring*

Legal Reasoning in Large Language Models. https://arxiv.org/abs/2308.11462.

Guttman, Louis. 1945. "A Basis for Analyzing Test-Retest Reliability." *Psychometrika* 10 (4): 255–82.

Gwet, Kilem Li. 2014. *Handbook of Inter-Rater Reliability*. 4th ed. Advanced Analytics.

Haberman, Shelby J. 1977. "Maximum Likelihood Estimates in Exponential Response Models." *The Annals of Statistics* 5 (5): 815–41.

Hambleton, Ronald K., and Hariharan Swaminathan. 1985. *Item Response Theory: Principles and Applications*. Kluwer-Nijhoff.

Hardt, Moritz, Nimrod Megiddo, Christos Papadimitriou, and Mary Wootters. 2016. "Strategic Classification." *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, 111–22.

Henderson, Charles R. 1953. "Estimation of Variance and Covariance Components." *Biometrics* 9 (2): 226–52.

Hendrycks, Dan, Collin Burns, Steven Basart, et al. 2021. "Measuring Massive Multitask Language Understanding." *Proceedings of the International Conference on Learning Representations (ICLR)*.

Hoeffding, Wassily. 1948. "A Class of Statistics with Asymptotically Normal Distribution." *The Annals of Mathematical Statistics* 19 (3): 293–325.

Holland, Paul W., and Howard Wainer. 1993. *Differential Item Functioning*. Lawrence Erlbaum Associates.

Holmstrom, Bengt, and Paul Milgrom. 1991. "Multitask Principal-Agent Analyses: Incentive Contracts, Asset Ownership, and Job Design." *Journal of Law, Economics, and Organization* 7: 24–52.

Horn, John L. 1965. "A Rationale and Test for the Number of Factors in Factor Analysis." *Psychometrika* 30 (2): 179–85.

Huang, Yuzhen, Yuzhuo Bai, Zhihao Zhu, et al. 2023. "C-Eval: A Multi-Level Multi-Discipline Chinese Evaluation Suite for Foundation Models." *Advances in Neural Information Processing Systems*.

Isbarov, Jafar, Arofat Akhundjanova, Mammad Hajili, et al. 2025. "TUMLU: A Unified and Native Language Understanding Benchmark for Turkic Languages." *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Vienna, Austria), 22816–38. https://doi.org/10.18653/v1/2025.acl-long.1112.

Islam, Pranab, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. 2023. *FinanceBench: A New Benchmark for Financial Question Answering*. https://arxiv.org/abs/2311.11944.

Jacovi, Alon, Ksenia Nezhurina, Fazl Barez, Timo Schick, and Thomas Scialom. 2023. "Stop Uploading Test Data in Plain Text: Practical Strategies for Mitigating Data Contamination by Evaluation Benchmarks." *arXiv Preprint arXiv:2305.10160*.

Jain, Naman, King Han, Alex Gu, et al. 2024. "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code." *arXiv Preprint arXiv:2403.07974*.

James, Stephen, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. 2020. "RLBench: The Robot Learning Benchmark & Learning Environment." *IEEE Robotics and Automation Letters* 5 (2): 3019–26. https://doi.org/10.1109/LRA.2020.2974707.

Jassem, Krzysztof, Michał Cieśiółka, Filip Graliński, et al. 2025. *LLMzSzŁ: A Comprehensive LLM Benchmark for Polish*. https://arxiv.org/abs/2501.02266.

Ji, Jiaming, Mickel Liu, Juntao Dai, et al. 2023. *BeaverTails: Towards Improved Safety Alignment of LLM via a Human-Preference Dataset*. https://arxiv.org/abs/2307.04657.

Jiang, Junfeng, Jiahao Huang, and Akiko Aizawa. 2025. "JMedBench: A Benchmark for Evaluating Japanese Biomedical Large Language Models." *Proceedings of the 31st International Conference on Computational Linguistics* (Abu Dhabi, UAE), 5918–35. https://aclanthology.org/2025.coling-main.395.

Jimenez, Carlos E., John Yang, Alexander Wettig, et al. 2023. "SWE-Bench: Can Language Models Resolve Real-World GitHub Issues?" *arXiv Preprint arXiv:2310.06770*.

Jin, Di, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2020. *What Disease Does This Patient Have? A Large-Scale Open Domain Question Answering Dataset from Medical Exams*. https://arxiv.org/abs/2009.13081.

Jin, Qiao, Bhuwan Dhingra, Zhengping Liu, William W. Cohen, and Xinghua Lu. 2019. "PubMedQA: A Dataset for Biomedical Research Question Answering." *arXiv Preprint arXiv:1909.06146*. https://arxiv.org/abs/1909.06146.

Kale, Neil, Chen Bo Calvin Zhang, Kevin Zhu, et al. 2025. "Reliable Weak-to-Strong Monitoring of LLM Agents." *arXiv Preprint arXiv:2508.19461*. https://arxiv.org/abs/2508.19461.

Kamenica, Emir, and Matthew Gentzkow. 2011. "Bayesian Persuasion." *American Economic Review* 101 (6): 2590–615.

Kane, Michael T. 2006. "Validation." In *Educational Measurement*, 4th ed., edited by Robert L. Brennan. Praeger.

Karabatsos, George. 2001. "The Rasch Model, Additive Conjoint Measurement, and New Models of Probabilistic Measurement Theory." *Journal of Applied Measurement* 2 (4): 389–423.

Kasai, Jungo, Yuhei Kasai, Keisuke Sakaguchi, Yutaro Yamada, and Dragomir Radev. 2023. *Evaluating GPT-4 and ChatGPT on Japanese Medical Licensing Examinations*. https://arxiv.org/abs/2303.18027.

Kiefer, Jack. 1959. "Optimum Experimental Designs." *Journal of the Royal Statistical Society: Series B* 21 (2): 272–319.

Kiela, Douwe, Max Bartolo, Yixin Nie, et al. 2021. "Dynabench: Rethinking Benchmarking in NLP." *arXiv Preprint arXiv:2104.14337*.

Kim, Seungone, Juyoung Suk, Ji Yong Cho, et al. 2025. *The BiGGen Bench: A Principled Benchmark for Fine-Grained Evaluation of Language Models with Language Models*. https://arxiv.org/abs/2406.05761.

Kingma, Diederik P., and Jimmy Ba. 2015. "Adam: A Method for Stochastic Optimization." *International Conference on Learning Representations (ICLR)*.

Koh, Jing Yu, Robert Lo, Lawrence Jang, et al. 2024. "VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks." *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 881–905. https://aclanthology.org/2024.acl-long.50.

Koller, Daphne, and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Koto, Fajri, Haonan Li, Sara Shatnawi, et al. 2024. "ArabicMMLU: Assessing Massive Multitask Language Understanding in Arabic." *Findings of the Association for Computational Linguistics: ACL 2024* (Bangkok, Thailand), 5622–40. https://doi.org/10.18653/v1/2024.findings-acl.334.

Krantz, David H., R. Duncan Luce, Patrick Suppes, and Amos Tversky. 1971. *Foundations of Measurement, Volume i: Additive and Polynomial Representations*. Academic Press.

Krippendorff, Klaus. 2011. "Computing Krippendorff's Alpha-Reliability." *Departmental Papers (ASC)*.

Kweon, Sunjun, Byungjin Choi, Gyouk Chu, et al. 2024. *KorMedMCQA: Multi-Choice Question Answering Benchmark for Korean Healthcare Professional Licensing Examinations*. https://arxiv.org/abs/2403.01469.

Kyngdon, Andrew. 2008. "The Rasch Model from the Perspective of the Representational Theory of Measurement." *Theory & Psychology* 18 (1): 89–109.

Laffont, Jean-Jacques, and Jean Tirole. 1986. "Using Cost Observation to Regulate Firms." *Journal of Political Economy* 94 (3): 614–41.

Lambert, Nathan, Valentina Pyatkin, Jacob Morrison, et al. 2024. "RewardBench: Evaluating Reward Models for Language Modeling." *arXiv Preprint arXiv:2403.13787*.

Lancaster, Tony. 2000. "The Incidental Parameter Problem Since 1948." *Journal of Econometrics* 95 (2): 391–413.

Li, Bohao, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. 2024. "SEED-Bench: Benchmarking Multimodal Large Language Models." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 13299–308. https://arxiv.org/abs/2307.16125.

Li, Haonan, Yixuan Zhang, Fajri Koto, et al. 2023. "CMMLU: Measuring Massive Multitask Language Understanding in Chinese." *arXiv Preprint arXiv:2306.09212*.

Li, Junlong, Fan Zhou, Shichao Sun, Yikai Zhang, Hai Zhao, and Pengfei Liu. 2024. "Dissecting Human and LLM Preferences." *arXiv Preprint arXiv:2402.11296*.

Li, Nathaniel, Alexander Pan, Anjali Gopal, et al. 2024. *The WMDP Benchmark: Measuring and Reducing Malicious Use with Unlearning*. https://arxiv.org/abs/2403.03218.

Li, Xuechen, Tianyi Zhang, Yann Dubois, et al. 2023. "AlpacaEval: An Automatic Evaluator of Instruction-Following Models." In *GitHub Repository*. Https://github.com/tatsu-lab/alpaca_eval[1]; GitHub.

Li, Yifan, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. 2023. "Evaluating Object Hallucination in Large Vision-Language Models." *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 292–305. https://aclanthology.org/2023.emnlp-main.20.

Liao, Thomas. 2025. *Why Do Eval Startups Fail?* Https://thomasliao.com/eval-startups[2].

Lightblue KK. 2024. *Tengu-Bench: LB-Bench, a Japanese LLM Evaluation Dataset*. Hugging Face Dataset. https://huggingface.co/datasets/lightblue/tengu_bench.

Lin, Bill Yuchen, Yuntian Deng, Khyathi Chandu, et al. 2024. *WildBench: Benchmarking LLMs with Challenging Tasks from Real Users in the Wild*. https://arxiv.org/abs/2406.04770.

Lin, Zi, Zihan Wang, Yongqi Tong, et al. 2023. "ToxicChat: Unveiling Hidden Challenges of Toxicity Detection in Real-World User-AI Conversation." *Findings of the Association for Computational Linguistics: EMNLP 2023* (Singapore), 4694–702. https://aclanthology.org/2023.findings-emnlp.311.

Linden, Wim J. van der. 2006. "Optimal Test Design." *Handbook of Statistics* 26: 575–95.

Liu, Jiawei, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation." *Thirty-Seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=1qvx610Cu7.

Liu, Junling, Peilin Zhou, Yining Hua, et al. 2023. "Benchmarking Large Language Models on CMExam–a Comprehensive Chinese Medical Exam Dataset." *arXiv Preprint arXiv:2306.03030*.

Liu, Yuan, Haodong Duan, Yuanhan Zhang, et al. 2024. "MMBench: Is Your Multi-Modal Model an All-Around Player?" *Proceedings of the European Conference on Computer Vision (ECCV)*. https://arxiv.org/abs/2307.06281.

Lord, Frederic M. 1970. "Some Test Theory for Tailored Testing." In *Computer-Assisted Instruction, Testing, and Guidance*. Harper; Row.

Lord, Frederic M., and Melvin R. Novick. 1968. *Statistical Theories of Mental Test Scores*. Addison-Wesley.

Lu, Pan, Hritik Bansal, Tony Xia, et al. 2024. "MathVista: Evaluating Mathematical Reasoning of Foundation Models in Visual Contexts." *Proceedings of the International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2310.02255.

Luce, R. Duncan, and John W. Tukey. 1964. "Simultaneous Conjoint Measurement: A New Type of Fundamental Measurement." *Journal of Mathematical Psychology* 1 (1): 1–27.

Luettgau, Lennart, Harry Coppock, Magda Dubois, Christopher Summerfield, and Cozmin Ududec. 2025. "HiBayES: A Hierarchical Bayesian Modeling Framework for AI Evaluation Statistics." *arXiv Preprint arXiv:2505.05602*.

Macina, Jakub, Nico Daheim, Ido Hakimi, Manu Kapur, Iryna Gurevych, and Mrinmaya Sachan. 2025. "MathTutorBench: A Benchmark for Measuring Open-Ended Pedagogical Capabilities of LLM Tutors." *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing* (Suzhou, China), 204–21. https://doi.org/10.18653/v1/2025.emnlp-main.11.

Malik, Saumya, Valentina Pyatkin, Sander Land, et al. 2025. *RewardBench 2: Advancing Reward Model Evaluation*. https://arxiv.org/abs/2506.01937.

Manheim, David, and Scott Garrabrant. 2018. "Categorizing Variants of Goodhart's Law." *arXiv Preprint arXiv:1803.04585*.

Marsman, Maarten, Denny Borsboom, Joost Kruis, et al. 2018. "An Introduction to Network Psychometrics: Relating Ising Network Models to Item Response Theory Models." *Multivariate Behavioral Research* 53 (1): 15–35.

Martı□nez-Plumed, Fernando, Pietro Baroni, W. René Carus, and Jose Hernandez-Orallo. 2024. "Item Response Theory in AI: Analysing Machine Learning Classifiers at the Instance Level." *Artificial Intelligence* 271: 18–42.

Masters, Geoff N. 1982. "A Rasch Model for Partial Credit Scoring." *Psychometrika* 47 (2): 149–74.

Maurya, Kaushal Kumar, Kv Aditya Srivatsa, Kseniia Petukhova, and Ekaterina Kochmar. 2025. "Unifying AI Tutor Evaluation: An Evaluation Taxonomy for Pedagogical Ability Assessment of LLM-Powered AI Tutors." *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1234–51. https://doi.org/10.18653/v1/2025.naacl-long.57.

Mazeika, Mantas, Long Phan, Xuwang Yin, et al. 2024. *HarmBench: A Standardized Evaluation Framework for Automated Red Teaming and Robust Refusal*. https://arxiv.org/abs/2402.04249.

McCullagh, Peter. 1980. "Regression Models for Ordinal Data." *Journal of the Royal Statistical Society: Series B (Methodological)* 42 (2): 109–27.

McDonald, Roderick P. 1999. *Test Theory: A Unified Treatment*. Lawrence Erlbaum Associates.

McGregor, Sean. 2021. "Preventing Repeated Real World AI Failures by Cataloging Incidents: The AI Incident Database." *Proceedings of the AAAI Conference on Artificial Intelligence* 35: 15458–63. https://doi.org/10.1609/aaai.v35i17.17817.

McLachlan, Geoffrey J., and Thriyambakam Krishnan. 2007. *The EM Algorithm and Extensions*. 2nd ed. Wiley-Interscience.

Merrill, Mike A., Alexander G. Shaw, Nicholas Carlini, et al. 2026. *Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces*. https://arxiv.org/abs/2601.11868.

Messick, Samuel. 1995. "Validity of Psychological Assessment: Validation of Inferences from Persons' Responses and Performances as Scientific Inquiry into Score Meaning." *American Psychologist* 50 (9): 741–49.

Michell, Joel. 2008. "Is Psychometrics Pathological Science?" *Measurement: Interdisciplinary Research and Perspectives* 6 (1–2): 7–24.

Mizrahi, Moran et al. 2024. "State of What Art? A Call for Multi-Prompt LLM Evaluation." *Transactions of the Association for Computational Linguistics*.

Murphy, Kevin P. 2022. *Probabilistic Machine Learning: An Introduction*. MIT Press.

National Highway Traffic Safety Administration. 2021. *Standing General Order on Crash Reporting*. U.S. Department of Transportation regulatory order. https://www.nhtsa.gov/laws-regulations/standing-general-order-crash-reporting.

Neyman, Jerzy, and Elizabeth L. Scott. 1948. "Consistent Estimates Based on Partially Consistent Observations." *Econometrica* 16 (1): 1–32.

Niklaus, Joel, Veton Matoshi, Pooja Rani, Andrea Galassi, Matthias Stürmer, and Ilias Chalkidis. 2023. "LEXTREME: A Multi-Lingual and Multi-Task Benchmark for the Legal Domain." *Findings of the Association for Computational Linguistics: EMNLP 2023*, 3016–54. https://doi.org/10.18653/v1/2023.findings-emnlp.200.

Nimo, Charles, Tobi Olatunji, Abraham Toluwase Owodunni, et al. 2025. "AfriMed-QA: A Pan-African, Multi-Specialty, Medical Question-Answering Benchmark Dataset." *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1948–73. https://doi.org/10.18653/v1/2025.acl-long.96.

Nocedal, Jorge, and Stephen J. Wright. 2006. *Numerical Optimization*. 2nd ed. Springer.

OECD. 2024. *OECD AI Incidents Monitor*. Online database. https://oecd.ai/en/incidents.

Oren, Yonatan, Nicole Meister, Niladri Chatterji, Faisal Ladhak, and Tatsunori B. Hashimoto. 2023. *Proving Test Set Contamination in Black Box Language Models*. https://arxiv.org/abs/2310.17623.

Pal, Ankit, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. "MedMCQA: A Large-Scale Multi-Subject Multi-Choice Dataset for Medical Domain Question Answering." In *Proceedings of the Conference on Health, Inference, and Learning*, edited by Gerardo Flores, George H Chen, Tom Pollard, Joyce C Ho, and Tristan Naumann, vol. 174. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v174/pal22a.html.

Pan, Alexander, Jun Shern Chan, Andy Zou, et al. 2023. "Do the Rewards Justify the Means? Measuring Trade-Offs Between Rewards and Ethical Behavior in the MACHIAVELLI Benchmark." *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 202: 26837–67. https://proceedings.mlr.press/v202/pan23a.html.

Passaglia, Sam. 2023. *Ranking Japanese LLMs with Rakuda*. Blog post, YuzuAI. https://yuzuai.jp/blog/rakuda.

Patil, Shishir G., Huanzhi Mao, Charlie Cheng-Jie Ji, et al. 2025. "The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models." *Forty-Second International Conference on Machine Learning*.

Patterson, H. Desmond, and Robin Thompson. 1971. "Recovery of Inter-Block Information When Block Sizes Are Unequal." *Biometrika* 58 (3): 545–54.

Pearl, Judea. 2009. *Causality: Models, Reasoning, and Inference*. 2nd ed. Cambridge University Press.

Peng, Sida, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot." *arXiv Preprint arXiv:2302.06590*. https://arxiv.org/abs/2302.06590.

Perdomo, Juan, Tijana Zrnic, Celestine Mendler-Dünner, and Moritz Hardt. 2020. "Performative Prediction." *Proceedings of the 37th International Conference on Machine Learning*, 7599–609.

Perez, Ethan, Sam Ringer, Kamilė Lukošiūtė, et al. 2022. "Red Teaming Language Models with Language Models." *arXiv Preprint arXiv:2202.03286*.

Perline, Richard, Benjamin D. Wright, and Howard Wainer. 1979. "The Rasch Model as Additive Conjoint Measurement." *Applied Psychological Measurement* 3 (2): 237–55.

Peters, Jonas, Dominik Janzing, and Bernhard Schölkopf. 2017. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press.

Phan, Long, Alice Gatti, Nathaniel Li, et al. 2026. "A Benchmark of Expert-Level Academic Questions to Assess AI Capabilities." *Nature* 649 (8099): 1139–46. https://doi.org/10.1038/s41586-025-09962-4.

Phuong, Mary et al. 2024. *Evaluating Frontier Models for Dangerous Capabilities*. https://arxiv.org/abs/2403.13793.

Polo, Felipe Maia, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. 2024. *tinyBenchmarks: Evaluating LLMs with Fewer Examples*. https://arxiv.org/abs/2402.14992.

Procaccia, Ariel D., Benjamin Schiffer, Serena Wang, and Shirley Zhang. 2025. "Metritocracy: Representative Metrics for Lite Benchmarks." *arXiv Preprint arXiv:2506.09813*.

Quiñonero-Candela, Joaquin, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. 2009. *Dataset Shift in Machine Learning*. MIT Press.

Raji, Inioluwa Deborah, Emily M. Bender, Amandalynne Paullada, Emily Denton, and Alex Hanna. 2021. "AI and the Everything in the Whole Wide World Benchmark." *Proceedings of the NeurIPS Track on Datasets and Benchmarks*.

Rasch, Georg. 1960. *Probabilistic Models for Some Intelligence and Attainment Tests*. Danish Institute for Educational Research.

Rashid, Muhammad Shihab, Christian Bock, Yuan Zhuang, et al. 2025. *SWE-PolyBench: A Multi-Language Benchmark for Repository Level Evaluation of Coding Agents*. https://arxiv.org/abs/2504.08703.

Rawles, Christopher, Sarah Clinckemaillie, Yifan Chang, et al. 2025. *AndroidWorld: A Dynamic Benchmarking Environment for Autonomous Agents*. https://arxiv.org/abs/2405.14573.

Revelle, William, and Richard E. Zinbarg. 2009. "Coefficients Alpha, Beta, Omega, and the Glb: Comments on Sijtsma." *Psychometrika* 74 (1): 145–54.

Ribeiro, Marco Tulio, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList." *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4902–12.

Rismani, Shalaleh et al. 2025. "Responsible AI Measures Dataset for Ethics Evaluation of AI Systems." *Nature Scientific Data*. https://www.nature.com/articles/s41597-025-06021-5.

Robins, James M., Andrea Rotnitzky, and Lue Ping Zhao. 1994. "Estimation of Regression Coefficients When Some Regressors Are Not Always Observed." *Journal of the American Statistical Association* 89 (427): 846–66.

Rofin, Mark, and Danila Mikhailov. 2023. "VOTE'N'RANK: Revision of Benchmarking with Social Choice Theory." *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, 670–86.

Romano, Antonio, Giuseppe Riccio, Mariano Barone, Marco Postiglione, and Vincenzo Moscato. 2025. "IMB: An Italian Medical Benchmark for Question Answering." *Proceedings of the Eleventh Italian Conference on Computational*

Linguistics (CLiC-it 2025), 995–1006. https://aclanthology.org/2025.clicit-1.93.

Romanov, Alexey, and Chaitanya Shivade. 2018. "Lessons from Natural Language Inference in the Clinical Domain." *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. https://arxiv.org/abs/1808.06752.

Salaudeen, Olawale, Anka Reuel, Ahmed Ahmed, et al. 2025. *Measurement to Meaning: A Validity-Centered Framework for AI Evaluation*. https://arxiv.org/abs/2505.10573.

Samejima, Fumiko. 1969. "Estimation of Latent Ability Using a Response Pattern of Graded Scores." *Psychometrika Monograph Supplement* 34 (4, Pt. 2): 1–97.

Schaeffer, Rylan, Joshua Kazdan, John Hughes, et al. 2025. "How Do Large Language Monkeys Get Their Power (Laws)?" *arXiv Preprint arXiv:2502.17578*.

Scheiblechner, Hartmann. 1995. "Isotonic Ordinal Probabilistic Models (ISOP)." *Psychometrika* 60 (2): 281–304.

Scheiblechner, Hartmann. 1999. "Additive Conjoint Isotonic Probabilistic Models (ADISOP)." *Psychometrika* 64 (3): 295–316.

Searle, Shayle R., George Casella, and Charles E. McCulloch. 1992. *Variance Components*. Wiley.

Segall, Daniel O. 1996. "Multidimensional Adaptive Testing." *Psychometrika* 61 (2): 331–54. https://doi.org/10.1007/BF02294343.

Shankar, Shreya et al. 2024. "Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences." *arXiv Preprint arXiv:2404.12272*.

Shavelson, Richard J., and Noreen M. Webb. 1991. *Generalizability Theory: A Primer*. Sage.

Shavrina, Tatiana, Alena Fenogenova, Anton Emelyanov, et al. 2020. "RussianSuperGLUE: A Russian Language Understanding Evaluation Benchmark." *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Online), 4717–26. https://doi.org/10.18653/v1/2020.emnlp-main.381.

Shen, Xu, Song Wang, Zhen Tan, et al. 2026. "FaithCoT-Bench: Benchmarking Instance-Level Faithfulness of Chain-of-Thought Reasoning." *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2510.04040.

Shimodaira, Hidetoshi. 2000. "Improving Predictive Inference Under Covariate Shift by Weighting the Log-Likelihood Function." *Journal of Statistical Planning and Inference* 90 (2): 227–44.

Singh, Shivalika et al. 2025. *The Leaderboard Illusion*. https://arxiv.org/abs/2504.20879.

Slattery, Peter, Alexander K. Saeri, Emily A. C. Grundy, et al. 2024. "The AI Risk Repository: A Comprehensive Meta-Review, Database, and Taxonomy of Risks from Artificial Intelligence." *arXiv Preprint arXiv:2408.12622*. https://arxiv.org/abs/2408.12622.

Sobol, Ilya M. 2001. "Global Sensitivity Indices for Nonlinear Mathematical Models and Their Monte Carlo Estimates." *Mathematics and Computers in Simulation* 55 (1–3): 271–80.

Son, Guijin, Hanwool Lee, Sungdong Kim, et al. 2024. *KMMLU: Measuring Massive Multitask Language Understanding in Korean*. https://arxiv.org/abs/2402.11548.

Spearman, Charles. 1910. "Correlation Calculated from Faulty Data." *British Journal of Psychology* 3 (3): 271–95.

Spence, Michael. 1973. "Job Market Signaling." *The Quarterly Journal of Economics* 87 (3): 355–74.

Srinivasa, Rakshith S, Zora Che, Chen Bo Calvin Zhang, et al. 2025. "TutorBench: A Benchmark to Assess Tutoring Capabilities of Large Language Models." *arXiv Preprint arXiv:2510.02663*. https://arxiv.org/abs/2510.02663.

Srivastava, Sanjana, Chengshu Li, Michael Lingelbach, et al. 2022. "BEHAVIOR: Benchmark for Everyday Household Activities in Virtual, Interactive, and Ecological Environments." *Proceedings of the 5th Conference on Robot Learning*, Proceedings of machine learning research, vol. 164: 477–90. https://proceedings.mlr.press/v164/srivastava22a.html.

Stevens, S. S. 1946. "On the Theory of Scales of Measurement." *Science* 103 (2684): 677–80.

Sugiyama, Masashi, Taiji Suzuki, and Takafumi Kanamori. 2012. *Density Ratio Estimation in Machine Learning*. Cambridge University Press.

Swaminathan, Hariharan, and Janice A. Gifford. 1982. "Bayesian Estimation in the Rasch Model." *Journal of Educational Statistics* 7 (3): 175–91. https://doi.org/10.3102/10769986007003175.

Syromiatnikov, Mykyta, Victoria Ruvinskaya, and Anastasiya Troynina. 2024. "ZNO-Eval: Benchmarking Reasoning Capabilities of Large Language Models in Ukrainian." *Proceedings of the x International Conference "Informatics. Culture. Technology."*, 185–91. https://arxiv.org/abs/2501.06715.

Thurstone, L. L. 1927. "A Law of Comparative Judgment." *Psychological Review* 34 (4): 273–86. https://doi.org/10.1037/h0070288.

Tianle Li*, Evan Frick, Wei-Lin Chiang*. 2024. *From Live Data to High-Quality Benchmarks: The Arena-Hard Pipeline*. https://lmsys.org/blog/2024-04-19-arena-hard/.

Tibshirani, Ryan J., Rina Foygel Barber, Emmanuel J. Candès, and Aaditya Ramdas. 2019. "Conformal Prediction Under Covariate Shift." *Advances in Neural Information Processing Systems* 32.

Trivedi, Harsh, Tushar Khot, Mareike Hartmann, et al. 2024. *AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents*. https://arxiv.org/abs/2407.18901.

Truong, Sang T., Yuheng Tu, Michael Hardy, et al. 2025. "Fantastic Bugs and Where to Find Them in AI Benchmarks." *arXiv Preprint*.

Truong, Sang, Yuheng Tu, Rylan Schaeffer, and Sanmi Koyejo. 2025. "Item Response Scaling Laws." *arXiv Preprint*.

Truong, Sang, Serena Wang, Nick Haber, and Sanmi Koyejo. 2025. "Strategic Evaluation: Incentivizing AI Capability Coverage with Private Benchmarks." *arXiv Preprint*.

Turpin, Miles, Julian Michael, Ethan Perez, and Samuel R. Bowman. 2023. "Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting." *Advances*

in Neural Information Processing Systems 36. https://proceedings.neurips.cc/paper_files/paper/2023/hash/ed3fea9033a80fea1376299fa7863f4a-Abstract-Conference.html.

Vapnik, Vladimir N. 1998. *Statistical Learning Theory*. Wiley.

Vehtari, Aki, Andrew Gelman, and Jonah Gabry. 2017. "Practical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and WAIC." *Statistics and Computing* 27 (5): 1413–32.

Verma, Sshubam, Mohammed Safi Ur Rahman Khan, Vishwajeet Kumar, Rudra Murthy, and Jaydeep Sen. 2024. *MILU: A Multi-Task Indic Language Understanding Benchmark*. https://arxiv.org/abs/2411.02538.

Vodrahalli, Kailas, Roxana Daneshjou, Tobias Gerstenberg, and James Zou. 2022. "Do Humans Trust Advice More If It Comes from AI? An Analysis of Human-AI Interactions." *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*. https://github.com/kailas-v/human-ai-interactions.

Vovk, Vladimir, Alex Gammerman, and Glenn Shafer. 2005. *Algorithmic Learning in a Random World*. Springer.

Wallach, Hanna, Meera Desai, A. Feder Cooper, et al. 2025. "Position: Evaluating Generative AI Systems Is a Social Science Measurement Challenge." *arXiv Preprint arXiv:2502.00561*.

Wang, Boxin, Weixin Chen, Hengzhi Pei, et al. 2024. *DecodingTrust: A Comprehensive Assessment of Trustworthiness in GPT Models*. https://arxiv.org/abs/2306.11698.

Wang, Peiyi et al. 2023. "Large Language Models Are Not Fair Evaluators." *arXiv Preprint arXiv:2305.17926*.

Wang, Serena, Michael I. Jordan, Katrina Ligett, and R. Preston McAfee. 2024. "Relying on the Metrics of Evaluated Agents." *arXiv Preprint arXiv:2402.14005*.

Wang, Xidong, Guiming Hardy Chen, Dingjie Song, et al. 2024. *CMB: A Comprehensive Medical Benchmark in Chinese*. https://arxiv.org/abs/2308.08833.

Wang, Yubo, Xueguang Ma, Ge Zhang, et al. 2024. *MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark*. https://arxiv.org/abs/2406.01574.

Wang, Zhilin, Yi Dong, Olivier Delalleau, et al. 2024. *HelpSteer2: Open-Source Dataset for Training Top-Performing Reward Models*. https://arxiv.org/abs/2406.08673.

Watanabe, Sumio. 2010. "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory." *Journal of Machine Learning Research* 11: 3571–94.

Weij, Teun van der, Felix Hofstätter, Ollie Jaffe, Samuel F. Brown, and Francis Rhys Ward. 2024. *AI Sandbagging: Language Models Can Strategically Underperform on Evaluations*. https://arxiv.org/abs/2406.07358.

White, Colin, Samuel Dooley, Manley Roberts, et al. 2025. *LiveBench: A Challenging, Contamination-Limited LLM Benchmark*. https://arxiv.org/abs/2406.19314.

Wright, Benjamin D., and Geofferey N. Masters. 1982. *Rating Scale Analysis: Rasch Measurement*. MESA Press.

Wright, Benjamin D., and Mark H. Stone. 1979. *Best Test Design*. MESA Press.

Xie, Qianqian, Weiguang Han, Zhengyu Chen, et al. 2024. *The FinBen: An Holistic Financial Benchmark for Large Language Models*. https://arxiv.org/abs/2402.12659.

Xie, Qianqian, Weiguang Han, Xiao Zhang, et al. 2023. *PIXIU: A Large Language Model, Instruction Data and Evaluation Benchmark for Finance*. https://arxiv.org/abs/2306.05443.

Xie, Tianbao, Danyang Zhang, Jixuan Chen, et al. 2024. *OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments*. https://arxiv.org/abs/2404.07972.

Xu, Qiantong, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. *On the Tool Manipulation Capability of Open-Source Large Language Models*. https://arxiv.org/abs/2305.16504.

Yin, Ziqi, Hao Wang, Kaito Horio, Daisuke Kawahara, and Satoshi Sekine. 2024. *JMMLU: Japanese Massive Multitask Language Understanding Benchmark*. GitHub repository, Waseda NLP. https://github.com/nlp-waseda/JMMLU.

Yu, Feiyang, Alex Moehring, Oishi Banerjee, Tobias Salz, Nikhil Agarwal, and Pranav Rajpurkar. 2024. "Heterogeneity and Predictors of the Effects of AI Assistance on Radiologists." *Nature Medicine*. https://www.nature.com/articles/s41591-024-02850-w.

Yu, Tianhe, Deirdre Quillen, Zhanpeng He, et al. 2020. "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning." *Proceedings of the Conference on Robot Learning*, Proceedings of machine learning research, vol. 100: 1094–100. https://proceedings.mlr.press/v100/yu20a.html.

Yu, Weihao, Zhengyuan Yang, Linjie Li, et al. 2024. "MM-Vet: Evaluating Large Multimodal Models for Integrated Capabilities." *Proceedings of the 41st International Conference on Machine Learning (ICML)*. https://arxiv.org/abs/2308.02490.

Yue, Xiang, Yuansheng Ni, Kai Zhang, et al. 2024. "MMMU: A Massive Multi-Discipline Multimodal Understanding and Reasoning Benchmark for Expert AGI." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://arxiv.org/abs/2311.16502.

Yüksel, Arda, Abdullatif Köksal, Lütfi Kerem Şenel, Anna Korhonen, and Hinrich Schütze. 2024. "TurkishMMLU: Measuring Massive Multitask Language Understanding in Turkish." *Findings of the Association for Computational Linguistics: EMNLP 2024* (Miami, Florida, USA), 7035–55. https://doi.org/10.18653/v1/2024.findings-emnlp.413.

Zan, Daoguang, Zhirong Huang, Ailun Yu, et al. 2024. *SWE-Bench-Java: A GitHub Issue Resolving Benchmark for Java*. https://arxiv.org/abs/2408.14354.

Zellers, Rowan, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. "HellaSwag: Can a Machine Really Finish Your Sentence?" *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4791–800.

Zhang, Andy K., Neil Perry, Riya Dulepet, et al. 2025. *Cybench: A Framework for Evaluating Cybersecurity Capabilities and Risks of Language Models*. https://arxiv.org/abs/2408.08926.

Zhang, Hugh et al. 2024. *A Careful Examination of Large Language Model Performance on Grade School Arithmetic*. https://arxiv.org/abs/2405.00332.

Zhang, Kaichen, Bo Lyu, Fanyi Wang, et al. 2025. "LMMs-Eval: Reality Check on the Evaluation of Large

Multimodal Models." *Findings of the Association for Computational Linguistics: NAACL 2025*. https://arxiv.org/abs/2407.12772.

Zhang, Shirley, and Moritz Hardt. 2024. "The Inherent Tradeoffs in LLM Benchmarks." *arXiv Preprint*.

Zhang, Zhexin, Leqi Lei, Lindong Wu, et al. 2024. *SafetyBench: Evaluating the Safety of Large Language Models*. https://arxiv.org/abs/2309.07045.

Zhao, Wenting, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. "WildChat: 1M ChatGPT Interaction Logs in the Wild." *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=Bl8u7ZRlbM.

Zheng, Lianmin, Wei-Lin Chiang, Ying Sheng, et al. 2023. "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena." *Advances in Neural Information Processing Systems*.

Zheng, Lucia, Neel Guha, Brandon R. Anderson, Peter Henderson, and Daniel E. Ho. 2021. "When Does Pretraining Help? Assessing Self-Supervised Learning for Law and the CaseHOLD Dataset of 53,000+ Legal Holdings." *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law* (New York, NY, USA), ICAIL '21, 159–68. https://doi.org/10.1145/3462757.3466088.

Zhou, Hongli, Hui Huang, Ziqing Zhao, et al. 2026. "Lost in Benchmarks? Rethinking Large Language Model Benchmarking with Item Response Theory." *Proceedings of the AAAI Conference on Artificial Intelligence* 40: 35085–93. https://doi.org/10.1609/aaai.v40i41.40814.

Zhou, Shuyan, Frank F. Xu, Hao Zhu, et al. 2024. *WebArena: A Realistic Web Environment for Building Autonomous Agents*. https://arxiv.org/abs/2307.13854.

Zhu, Banghua, Evan Frick, Tianhao Wu, Hanlin Zhu, and Jiantao Jiao. 2023. *Starling-7B: Improving LLM Helpfulness & Harmlessness with RLAIF*[3].

Zhu, Fengbin, Wenqiang Lei, Youcheng Huang, et al. 2021. *TAT-QA: A Question Answering Benchmark on a Hybrid of Tabular and Textual Content in Finance*. https://arxiv.org/abs/2105.07624.

Zhuo, Terry Yue, Minh Chien Vu, Jenny Chim, et al. 2025. *BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions*. https://arxiv.org/abs/2406.15877.

Zumbo, Bruno D. 1999. *A Handbook on the Theory and Methods of Differential Item Functioning (DIF)*. National Defense Headquarters.

[1] https://github.com/tatsu-lab/alpaca_eval
[2] https://thomasliao.com/eval-startups
[3]

Index

analytic flexibility, *see also* p-hacking
anonymization, *see also* de-identification
APA, *see* American Psychological Association (APA)

blinding, *see* masking

CDI, *see* Communicative Development Inventory

Cohen's d, *see also* standardized mean difference (SMD)

DAG, *see* directed acyclic graph (DAG)
de-identification, *see also* anonymization

p-hacking, *see also* analytic flexibility

standardized mean difference (SMD), *see also* Cohen's d