AI Measurement Science

AI Measurement Science

A Science of Knowing Where AI Thrives, Where It Breaks, and How to Respond

Sang T. Truong, and Sanmi Koyejo

Stanford University
Stanford, CA

Table of Contents

Preface

Every claim about AI progress—better reasoning, broader capability, safer behavior—rests on some act of measurement. But how good are our measurements, really? Benchmarks multiply faster than anyone can understand them, leaderboards rank models on scores without scales, and a system's average accuracy tells us little about *why* it succeeds or *where* it breaks. We have plenty of numbers. What we lack is a science behind them.

This book introduces the foundations and practical tools of measurement science for AI evaluation. Instead of treating evaluation as a purely engineering task—collect a dataset, compute a metric, publish a number—we present it as an *inference* problem, drawing on ideas from psychometrics, statistics, and the broader science of measurement. Throughout, we emphasize not only the methods themselves but also their assumptions, limitations, and the conditions under which they can and cannot be applied—understanding when an evaluation fails is as important as understanding when it works. By the end of this book, readers will be equipped with the key concepts and tools needed to design evaluation systems that are reliable, valid, and interpretable.

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 claims about AI systems rest on solid scientific ground.

Stanford, 2025, Truong & Koyejo

Structure of this book

The book opens with a survey of the evaluation data landscape, followed by three parts which introduce predictive measurement models, develop reliability and validity analysis, and discuss design, governance, and applications.

Chapter 1 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. It introduces the `torch_measure` toolkit for loading and analyzing benchmark data.

Part I: Measurement as Predictive Modeling

Chapter 2 lays the mathematical groundwork for the rest of the book. It covers probabilistic models for AI evaluation data—including Item Response Theory (Rasch, 2PL, 3PL), Bradley-Terry models for pairwise comparisons, and factor models for multidimensional ability. 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 factor model that maps external features to latent parameters for cold-start prediction of unseen models and items.

Chapter 3 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. It also develops generalization experiments with various masking schemes to evaluate the robustness and transferability of learned factor models.

Chapter 4 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.

Part II: Measurement Reliability and Validity

Chapter 5 addresses the question of signal versus noise in AI measurement. It covers Classical Test Theory, Generalizability Theory for decomposing multiple sources of error, and practical tools for assessing and improving the reliability of evaluation protocols—including LLM-as-a-judge designs and agentic measurement.

Chapter 6 asks whether our evaluations measure what they claim to measure. It covers content, criterion, and construct validity; diagnostic tools for detecting benchmark contamination, differential item functioning, and construct-irrelevant variance; and principled approaches to instrument construction and revision, including the use of synthetic data.

Chapter 7 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.

Part III: Design, Governance, and Applications

Chapter 8 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 9 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.

Conclusion

Chapter 10 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

Chapter 2 is foundational to all of the book, so it is part of all reading pathways.

For **practitioners and applied AI researchers**, we recommend Chapter 2, Chapter 3, Chapter 4, and Chapter 8. This sequence covers the core modeling, estimation, and design methods needed to build and improve evaluation systems.

For **researchers interested in evaluation quality**, we recommend Chapter 2, Chapter 5, Chapter 6, and Chapter 7. This pathway focuses on when and why evaluations fail, and how to diagnose and address those failures.

For those interested in the **societal and governance dimensions** of AI evaluation, we recommend Chapter 2, Chapter 8, and Chapter 9.

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:

- 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 Chapters 1–3.
- Bishop (2006) — Thorough coverage of factor analysis, EM algorithms, and Bayesian methods, all of which appear throughout this book.
- 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 Chapters 4–5.

- 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.

Citation

Thanks for reading our book! We hope you find it useful in your research and teaching.

1 The Data Landscape

Intended Learning Outcomes

By the end of this chapter, you 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)

1.1 *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**: $Y_{ij} = \texttt{NaN}$ (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)
Code generation	LLMs	Programming problems	Binary or pass@1	BigCodeBench (153 × 1,140)
Agent tasks	Agent + scaffold	Multi-step episodes	Binary	SWE-bench (134 × 500)
Function calling	LLMs	API call specifications	Binary	BFCL (93 × 4,751)
Web navigation	Agent + scaffold	Web interaction tasks	Binary	WebArena (14 × 812)
Terminal tasks	Agent + scaffold	System admin tasks	Continuous	Terminal-Bench (128 × 89)
Code reasoning	LLMs	Input/output prediction	Continuous	CRUXEval (38 × 800)

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.

1.1.1 *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) 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 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.

1.2 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**.

1.2.1 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), LiveBench, and HLE 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+), competitive programming (LiveCodeBench: 1,055 problems from AtCoder, LeetCode, and CodeForces), library-aware code generation (BigCodeBench: 1,140 tasks using 139 APIs), code reasoning (CRUX-Eval: 800 input/output prediction problems), code editing (EditBench: 540 editing tasks), and full software engineering (SWE-bench: resolving real GitHub issues). 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), mobile device automation (AndroidWorld: 116 tasks), desktop interaction (OSWorld), multi-app coordination (AppWorld: 24 tasks), terminal operations (Terminal-Bench: 89 system administration tasks), and security challenges (CyBench: 40 CTF tasks). 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) and ToolBench 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. {#sec-redteaming} This category spans cybersecurity challenges (CyBench: 40 CTF tasks), security of tool-using agents (AgentDojo: 949 security items), 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) provides a standardized red teaming benchmark; BeaverTails (334K prompts with fine-grained safety annotations) and DecodingTrust (243K prompts across 8 trustworthiness dimensions) offer larger-scale evaluation. SafetyBench (11,435 MCQs across 7 safety categories) tests safety knowledge in a traditional MCQ format, while WMDP (3,668 MCQs on biosecurity, cybersecurity, and chemical security) 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, AlpacaEval (805 instructions), MT-Bench, Arena Hard, WildBench, and the Chatbot Arena (140K+ comparisons) 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 1.4.

1.2.2 *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; CRUXEval reports pass@1 from 10 samples. Continuous responses carry richer information than binary responses and motivate extensions like Beta-IRT (Chapter 3).

Ordinal responses ($Y_{ij} \in \{1, 2, \dots, L\}$) 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 1.4.

1.2.3 *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, LiveCodeBench) or generate items adversarially (DynaBench). LiveCodeBench draws from ongoing programming competitions, ensuring that items postdate model training cutoffs. 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, WebArena, and Terminal-Bench 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.

1.2.4 *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/editbench[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/beavertails[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/SafetyBench[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	IgakuQA	5 × 1,471	Bin	MCQ; Japanese medical licensing exams (2018–2022)	2023-03	jungokasai/IgakuQA[53]
54	LawBench	51 × 9,000	Bin	Exact match; 20 Chinese legal tasks, zero-shot	2023-09	open-compass/LawBench[54]
55	FinanceBench	16 × 150	Bin	Expert-graded; SEC filing QA	2023-11	patronus-ai/financebench[55]
84	LexEval	38 × 14,147	Con	Rubric; 23 Chinese legal tasks (NeurIPS 2024)	2024-09	CSHaitao/LexEval[56]
78	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.

1.2.5 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, HLE, LiveBench) grouping together and multilingual benchmarks (C-Eval, CMMLU, KMMLU, AfriEval, IberBench) spreading along a language axis. **Code benchmarks** (BigCodeBench, EvalPlus, CRUXEval) occupy a distinct region, reflecting the semantic difference between natural language questions and programming tasks. **Software engineering benchmarks** (SWE-bench 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 using the factor models in Chapter 2.

1.3 *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.

1.3.1 *Regional Leaderboards and Benchmarks*

Multiple evaluation efforts target specific linguistic and cultural communities:

Benchmark	Focus	Coverage
C-Eval, CMMLU	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 6 are directly applicable.

1.3.2 Multilingual Software Engineering

The multilingual dimension extends beyond language tasks. SWE-bench Multilingual and SWE-bench Java test software engineering in languages other than Python, revealing that "coding ability" as measured by Python-only benchmarks may not transfer. SWE-PolyBench 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?

1.4 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), AlpacaEval, MT-Bench, Arena Hard, WildBench, and preference datasets (HH-RLHF, UltraFeedback, HelpSteer2, Nectar, SHP-2) all produce comparison data.

1.4.1 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 2.2.3) 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.

1.4.2 Reward Model Benchmarks

RewardBench and RewardBench 2 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 and BigGen go further, decomposing preference judgments into multiple criteria (helpfulness, safety, coherence, creativity), producing multi-trait response data suitable for multidimensional measurement models.

1.5 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.

1.5.1 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 (**?@sec-many-facet**). 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.

1.5.2 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 (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 (METR 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.

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.

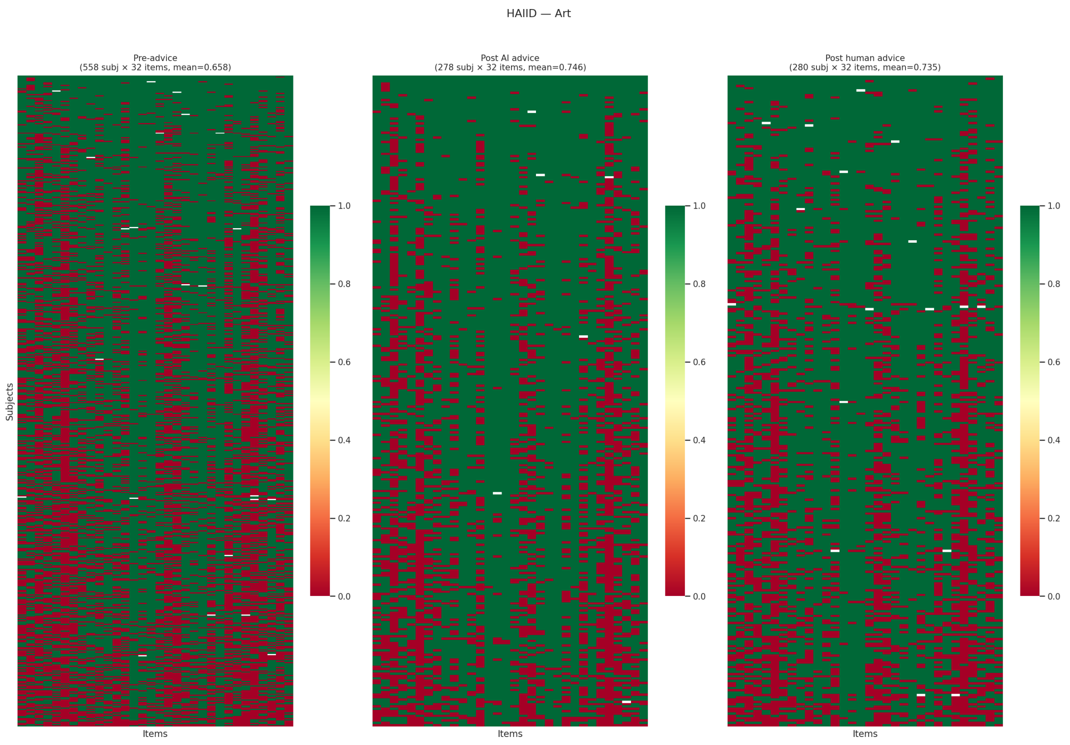

Figure 1.1
Paired response matrices from the HAIID art classification domain. Each panel shows a participant × item response matrix (green = correct, red = incorrect) under a different condition. **Left**: Pre-advice responses (558 participants × 32 items, mean accuracy 0.658). **Center**: Post-advice 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.

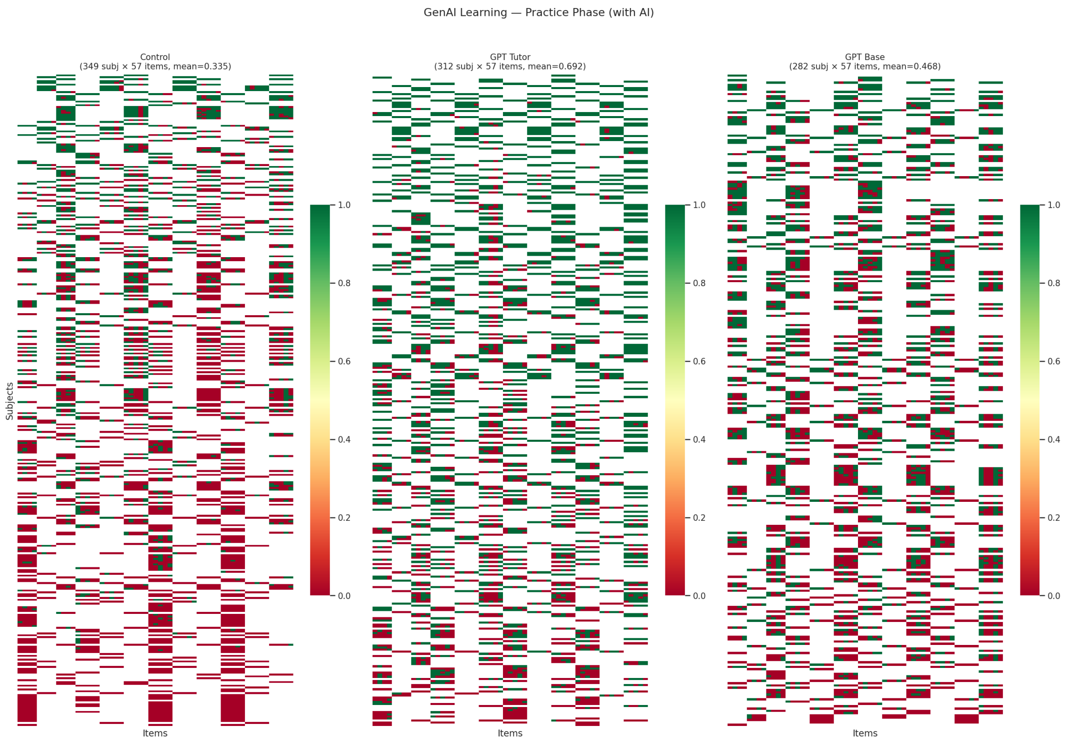

Figure 1.2
Paired response matrices from the GenAI Learning study, practice phase. Each panel shows a student × problem response matrix (green = correct, red = incorrect) under a different treatment arm. **Left**: Control group (349 students, mean 0.335). **Center**: GPT Tutor with pedagogical guardrails (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.

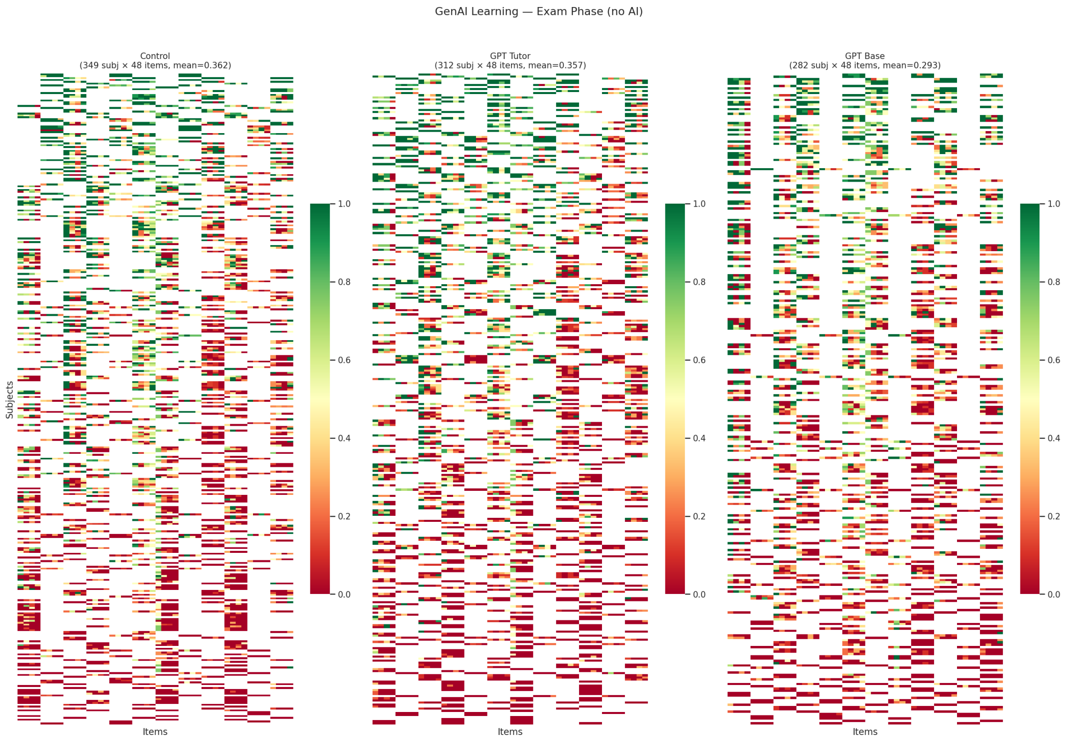

Figure 1.3
The same three treatment arms on a subsequent exam taken *without AI assistance*. **Left**: Control (mean 0.362). **Center**: GPT Tutor (mean 0.357). **Right**: GPT Base (mean 0.293). The visual similarity of the control and GPT Tutor panels — and the increased red in the GPT Base panel — shows that AI-inflated practice performance did not transfer to durable learning.

1.5.3 *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 (**?@sec-dif**) 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.

1.6 *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.

1.6.1 *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) is now curated with full per-question per-model predictions. LegalBench (162 English legal reasoning tasks, 12K+ items), LEXTREME (24 European languages), and CaseHOLD (53K US case law items) 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) is now curated. FinBen/PIXIU covers 36 datasets across 24 tasks using the `lm-evaluation-harness` framework. FinQA (8,281 numerical reasoning questions), ConvFinQA (multi-turn financial QA), and TAT-QA (hybrid tabular + textual QA) have public datasets but no cross-model prediction data.

Healthcare. AfriMedQA (Pan-African, 30 models × 6,910 items) is curated. MedQA (12,723 USMLE-style questions), MedMCQA (194K Indian medical exam questions), and PubMedQA (1,000 expert-labeled questions) all have public question datasets. Multilingual medical benchmarks include CMB/CMExam (Chinese), KorMedMCQA (Korean), JMedBench/IgakuQA (Japanese), MedExpQA (4 languages), and IMB (Italian, 25K+ items). 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), UltraFeedback (17 models × 64K prompts), AlpacaEval, and WildBench are curated. MT-Bench (80 questions) has item content but only pairwise human judgments, not per-model absolute scores. The Chatbot Arena (33K conversations with pairwise preferences) 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, SEED-Bench, MME, POPE, MM-Vet, MMMU, MathVista, Video-MME, and many more). This is effectively a massive collection of response matrices for multimodal evaluation, ready for IRT analysis. The `lmms-eval` framework produces per-item predictions that could be curated similarly. Our collection's limited multimodal coverage (only VisualWebArena) is a curation gap, not a data availability gap.

1.6.2 *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** (9 Turkic languages, 14 models, structured JSON with question + model output per item), **Rakuda** and **Tengu-Bench** (Japanese, ~558 models via the Shaberi framework), and **IgakuQA** (Japanese medical, 5 baselines). These are ready for immediate curation.

Second, many benchmarks publish questions but not model predictions. JMMLU (Japanese), MILU (11 Indic languages), TurkishMMLU, LLMzSzL (Polish), and ZNO (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, Russian SuperGLUE) 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 (Korean), ArabicMMLU (native Arabic questions sourced from regional exams), LawBench (Chinese legal system), and TUMLU (Turkic cultural knowledge) are positive examples; translated MMLU variants are not.

1.6.3 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, C-Eval, KMMLU), few evaluate AI *in educational settings*. MathTutorBench, TutorBench, and MRBench 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, RLBench, Meta-World), 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, 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 (AIID, 1,404 incidents with CSET harm taxonomy), the OECD AI Incidents Monitor (14,000+ incidents), and the MIT AI Risk Repository (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 (4.8M real ChatGPT conversations with per-message moderation flags) and LMSYS ToxicChat (10K labeled production prompts) effectively provide conversation × safety-category binary matrices from deployed systems. NVIDIA Aegis 2.0 (33K interactions × 23 hazard categories) provides a similar structure. **Regulatory monitoring data** — notably the NHTSA Standing General Order crash reports (1,700+ self-driving incidents as downloadable CSVs) and California DMV autonomous vehicle disengagement reports (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, Zaharia, and Zou'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.

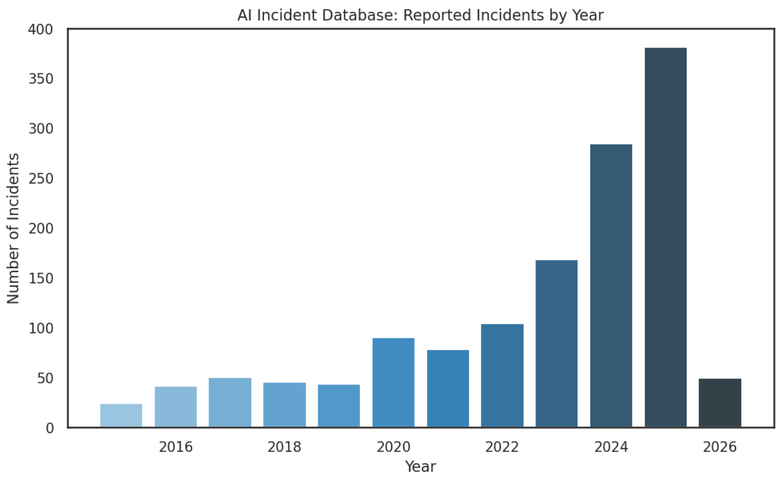

Figure 1.4
Reported AI incidents by year from the AI Incident Database (AIID). The exponential growth — from 24 incidents in 2015 to 381 in 2025 — reflects both increasing deployment of AI systems and increasing reporting infrastructure. The 2026 bar is partial (data through March 2026).

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

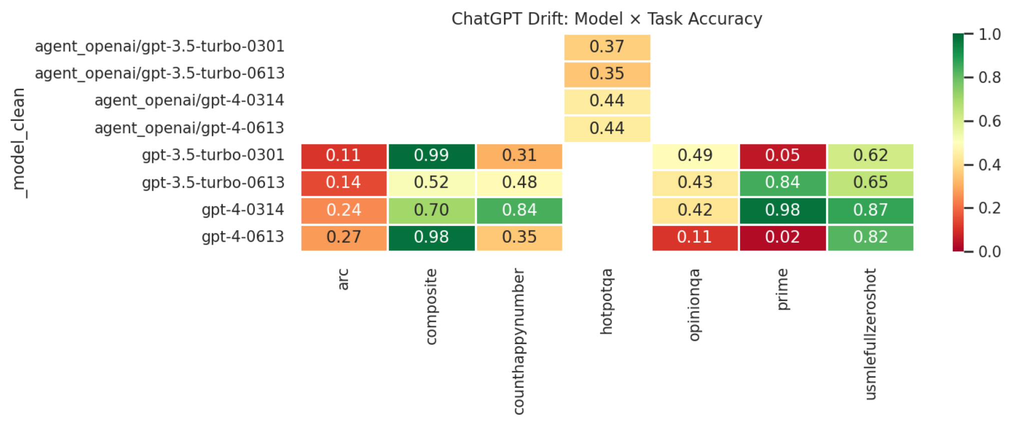

Figure 1.5
ChatGPT drift: model × task accuracy heatmap showing performance degradation across API versions. GPT-3.5-turbo's accuracy on composite tasks dropped from 0.99 (March 2023 version) to 0.14 (June 2023 version). GPT-4 shows similar but less dramatic degradation. This demonstrates that response matrices are not stationary — the same model can produce different responses to the same items over time.

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 (2.14M RL training transcripts with alignment-faking labels, where 34.6% of samples show alignment faking), Scale AI's Monitor Red Teaming dataset (MRT, thousands of agent trajectories attempting to evade monitors via lying and manipulation), BELLS (labeled execution traces across hallucination, jailbreak, and prompt injection categories), and the MACHIAVELLI benchmark (572K game scenes with dense ethical violation annotations across 25+ categories). CoT faithfulness datasets — FaithCoT-Bench (1,000+ annotated trajectories with step-level faithfulness labels) and Turpin et al.'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) evaluates reward models, and the PSN-IRT project (AAAI 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 6 remain rare. Applying the reliability and validity tools from Chapter 5 and Chapter 6 to evaluation methods themselves — not just the systems being evaluated — is an important frontier.

1.7 *Data Quality in Practice*

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

1.7.1 *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.

1.7.2 *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. SWE-bench has entries like "SWE-Agent + Claude Sonnet 4" and "OpenHands + Claude Sonnet 4" — same model, different scaffolds, substantially different performance. 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.

1.7.3 *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. But this introduces a new challenge: the item pool changes over time, making longitudinal comparisons difficult without equating procedures.

1.7.4 *Missing Data Patterns*

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

- **Temporal missingness**: older models evaluated on fewer items (LiveCodeBench)
- **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 3.7 provide a framework for testing how well models handle different missingness patterns.

1.8 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.

1.8.1 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}")
```

1.8.2 Inspecting the Response Matrix

```python
# 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()}")
```

1.8.3 Fitting a Measurement Model

```python
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

```

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

1.8.4 *Comparing Benchmarks*

```python
# 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}")
```

1.8.5 *Loading Paired Response Matrices*

```python
# 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 2 introduces the probabilistic models that decompose these matrices into interpretable latent parameters. Chapter 3 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.

1.9 Discussion Questions

1. **The subject identity problem**. For SWE-bench, the top-performing entry uses a proprietary scaffold with Claude Opus 4.5. 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 would you design a study 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. 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. 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 would you use IRT equating procedures to 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 would you detect and quantify transitivity violations in Chatbot Arena data? 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) while others have systematic gaps (LiveCodeBench: 88.8%). How does the missingness pattern affect the validity of IRT ability estimates? Under what conditions would you trust ability estimates from a sparse matrix?

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 would you design an evaluation framework that distinguishes 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?

1.10 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) 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 (METR 2025), which produced the surprising finding that experienced developers were slower with AI assistance. The Collab-CXR dataset (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 (Li et al. 2023) provide Chinese-language evaluation. KMMLU provides Korean-language evaluation. IberBench covers Iberian languages. AfriEval and AfriMedQA address African languages and healthcare contexts. SIB-200 provides classification across 200+ languages. HELM Multilingual (Bommasani, Liang, and Lee 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 Terry (1952). 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).

I

Measurement as Predictive Modeling

2 Foundations of Measurement

Intended Learning Outcomes

By the end of this chapter, you 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** why the Rasch model holds a special status as "the measurement model" through the sufficiency of sum scores, specific objectivity, and test-free measurement.
3. **Derive** the sufficiency theorem for the Rasch model and explain its implications for AI benchmark evaluation.
4. **Compare** the prescriptive (Rasch school) and descriptive (general IRT) approaches to measurement and articulate when each is appropriate.
5. **Trace** the historical development from Thurstone (1927) through Rasch (1960) to modern network psychometrics.
6. **Connect** classical measurement concepts (reliability, validity, dimensionality) to contemporary AI benchmark evaluation.
7. **Apply** measurement theory to analyze whether AI benchmarks satisfy the requirements for scientific measurement.
8. **Implement** basic IRT models in Python and visualize item characteristic curves.
9. **Evaluate** the assumptions underlying AI leaderboards and identify potential violations of measurement principles.

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), a_j (discrimination), c_j (guessing), Y_{ij} (binary response), and $\sigma(\cdot)$ (logistic sigmoid). See **?@sec-notation** for the complete notation reference.

2.1 The Measurement Problem in AI

Consider the following scenario: You have evaluated 100 language models on a benchmark consisting of 1,000 multiple-choice questions. Each model either answers each question correctly (1) or incorrectly (0), producing a 100×1000 binary response matrix Y. You compute each model's accuracy—the proportion of correct answers—and rank the models accordingly.

Have you *measured* anything?

The answer is not as obvious as it might seem. You have certainly *scored* the models: you assigned numbers to them based on their performance. But measurement, in the scientific sense, requires more than assigning numbers. It requires that those numbers correspond to some underlying property—a *latent construct*—in a principled way.

2.1.1 Scoring vs. Measuring

The distinction between scoring and measuring is fundamental to understanding why AI evaluation needs measurement science. Consider an analogy from physics: if you measure the temperature of water with a mercury thermometer, the height of the mercury column is a *score*—a number you can read off the instrument. But you trust this score as a *measurement* of temperature because you understand the relationship between mercury expansion and thermal energy.

In AI evaluation, we often have scores without this deeper understanding. When GPT-4 achieves 86% accuracy on MMLU and Claude achieves 84%, we cannot immediately conclude that GPT-4 has more "intelligence" or "capability" than Claude. Several questions must be answered first:

1. **What latent construct does MMLU measure?** Is it general intelligence, factual knowledge, test-taking ability, or something else entirely?
2. **Is the construct unidimensional?** Can model performance be characterized by a single number, or do different questions tap into different capabilities?
3. **Are the scores comparable across different test conditions?** Would the ranking change if we used different questions from the same domain?
4. **What is the measurement error?** How much of the score difference reflects true differences in capability versus noise?

These questions have been central to psychology and education for over a century. The field of *psychometrics* developed sophisticated tools—Item Response Theory, factor analysis, validity frameworks—precisely to address them. AI evaluation is now confronting the same fundamental challenges.

2.1.2 The Response Matrix

The basic data structure in measurement is the *response matrix* $Y \in \{0,1\}^{N \times M}$, where:

- Each row $i \in \{1, \dots, N\}$ represents a *test taker* (in AI: a model)
- Each column $j \in \{1, \dots, M\}$ represents an *item* (in AI: a benchmark question)
- Each entry $Y_{ij} \in \{0,1\}$ indicates whether test taker i answered item j correctly

$$Y = \begin{pmatrix} Y_{11} & Y_{12} & \cdots & Y_{1M} \\ Y_{21} & Y_{22} & \cdots & Y_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{N1} & Y_{N2} & \cdots & Y_{NM} \end{pmatrix}$$

The naive approach to evaluation computes row means (model accuracies) and ranks models accordingly. But the response matrix contains far more information than these marginal statistics. The *pattern* of responses—which models succeed on which questions—reveals structure that aggregate scores obscure.

```python
#| 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,
})
```

```python
#| label: response-matrix-visualization
#| autorun: true
#| fig-cap: "Response matrix from a language model evaluation. Rows are models (sorted
↪  by total score), columns are questions (sorted by difficulty). The diagonal
↪  structure suggests underlying ability and difficulty parameters."

# Simulate a response matrix with Rasch model structure
N, M = 50, 200  # 50 models, 200 questions

```

```python
# Generate latent abilities and difficulties
theta = np.random.normal(0, 1, N)  # model abilities
beta = np.random.normal(0, 1.5, M)  # question difficulties

# Generate responses via Rasch model
prob = 1 / (1 + np.exp(-(theta[:, None] - beta[None, :])))
Y = (np.random.random((N, M)) < prob).astype(int)

# Sort by row and column sums
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]

# Plot
fig, ax = plt.subplots(1, 1)
ax.imshow(Y_sorted, aspect='auto', cmap='Blues', interpolation='nearest')
ax.set_xlabel('Questions (sorted by difficulty)')
ax.set_ylabel('Models (sorted by ability)')
ax.set_title('Sorted Response Matrix')

plt.show()
```

When we sort the response matrix by row sums (model abilities) and column sums (item difficulties), a characteristic diagonal structure emerges. High-ability models answer most questions correctly; easy questions are answered correctly by most models. This structure is not guaranteed—it depends on the data satisfying certain assumptions—but when present, it suggests that a simple latent variable model may adequately describe the data.

2.1.3 Why AI Evaluation Needs Measurement Science

The problems facing AI evaluation today mirror those that psychology confronted in the early 20th century:

1. **Construct definition**: What does it mean to measure "reasoning" or "common sense"? Psychology developed validity frameworks to address this question.
2. **Test bias**: Are some benchmark questions unfair to certain models due to training data or architecture? Educational testing developed differential item functioning (DIF) analysis.
3. **Score comparability**: Can we compare models evaluated on different benchmark subsets? Psychometrics developed test equating methods.
4. **Efficiency**: How can we evaluate models with fewer questions? Computerized adaptive testing (CAT) emerged from IRT.
5. **Reliability**: How stable are our rankings under different conditions? Test-retest reliability and standard error of measurement quantify this.

The tools developed in psychometrics are not merely analogies—they are directly applicable to AI evaluation. The response matrix from an LLM benchmark has the same structure as the response matrix from a standardized test. The statistical models that describe human test performance can describe AI benchmark performance.

The Central Claim of AIMS

AI benchmarks are tests in the psychometric sense. The methods developed over a century of educational and psychological measurement—Item Response Theory, factor analysis, validity frameworks—apply directly to AI evaluation. Understanding and applying these methods is essential for trustworthy AI measurement.

2.1.4 *Evaluation Datasets Used in This Book*

Throughout this book, we work with several large-scale evaluation corpora that represent distinct yet complementary perspectives on measuring model behavior. These datasets provide the empirical foundation for our analyses.

HELM Benchmark Suite. We use **22 datasets** drawn from **5 HELM repositories**—*Classic*, *Lite*, *AIR-Bench*, *Thai Exam*, and *MMLU*—encompassing both *capability* and *safety* measurements. In total, this collection includes **172 test takers** (models) and **217,268 questions**. We focus on responses that can be graded dichotomously (correct/incorrect), as is the case for most benchmarks through metrics such as *exact match* or *equivalent indicator*. To ensure stable estimation, we remove duplicate questions, those with identical response patterns, or with fewer than 30 test takers; exclude test takers with fewer than 30 total responses; and treat unattempted questions as missing values.

Open LLM Leaderboard. We use data from the **Open LLM Leaderboard** (Hugging Face, 2025), a public benchmarking platform that evaluates open large language models on a standardized suite of academic and practical tasks. The dataset spans models submitted between **2022 and 2025**, covering parameter scales from small models (<5B parameters) to frontier systems (>140B parameters). In total, it includes **4,416 distinct language models**, each evaluated on **21,176 benchmark questions** from six suites: MMLU-Pro, OpenLLM-Math, MUSR, BBH, IFEval, and GPQA.

LMarena Preference Data. In addition to correctness-based evaluation, we incorporate **pairwise preference data** from the **LMarena dataset**, which provides human or automated judgments of relative model quality. Each example corresponds to a prompt presented to two competing models, with an annotation indicating which response is preferred. The dataset includes **211,728 unique prompts**, **3,779 unique model pairs**, and **179 distinct models**. These preference judgments provide a complementary view focusing on *relative comparisons* rather than absolute correctness.

Agent Leaderboard. We include **Agent Leaderboard data** from Galileo AI, which evaluates the *agentic performance* of large language models across tool-use and reasoning scenarios. This dataset contains approximately **34,700 rows**, where each row corresponds to a question, the model's response, and a numerical score judged by GPT-4. The evaluation covers multiple agentic subjects with roughly 100 questions each, including approximately **40 distinct models** such as Gemini-2.5, Claude-3.5, GPT-4.1/4.5, Llama-4, and Qwen-2.5.

Together, these four sources enable unified modeling of *accuracy*, *preference*, and *agency* within a shared latent-factor evaluation framework.

2.2 Probabilistic Models for Measurement

Measurement requires a model connecting observable responses to latent constructs. This section surveys the major families of probabilistic models used in measurement: Item Response Theory, factor models, paired comparison systems, network models, and hierarchical models. These models provide the statistical machinery for extracting latent variables from response data.

2.2.1 Item Response Theory

Item Response Theory (IRT) models the probability of a correct response as a function of person ability and item characteristics. The key insight is that both persons and items can be characterized by parameters on a common scale.

2.2.1.1 The Rasch Model (1PL)

The simplest IRT model is the **Rasch model**, also called the one-parameter logistic (1PL) model:

Definition: Rasch Model

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

where:

- $\theta_i \in \mathbb{R}$ is the ability of person i
- $\beta_j \in \mathbb{R}$ is the difficulty of item j
- $\sigma(\cdot)$ is the logistic sigmoid function

The model has an elegant interpretation: the probability of success depends only on the *difference* between ability and difficulty. When $\theta_i = \beta_j$, the probability is exactly 0.5—the person has a 50% chance of answering correctly. When $\theta_i > \beta_j$, the probability exceeds 0.5; when $\theta_i < \beta_j$, it falls below 0.5.

The function $P(\theta) = \sigma(\theta - \beta)$ is called the **Item Characteristic Curve (ICC)**. It describes how the probability of success changes with ability for a fixed item.

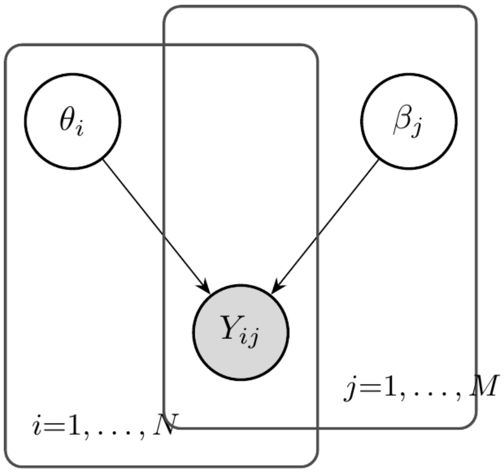

Figure 2.1
Plate diagram for the Rasch model. Shaded nodes are observed; open nodes are latent. Plates indicate replication over persons (i) and items (j).

```
#| label: icc-rasch
#| autorun: true
#| fig-cap: "Item Characteristic Curves for Rasch model items with different
↪  difficulties. All curves have the same shape (slope), differing only in their
↪  location."

import numpy as np
import matplotlib.pyplot as plt

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

theta = np.linspace(-4, 4, 200)
difficulties = [-2, -1, 0, 1, 2]

plt.figure()
for beta in difficulties:
    prob = sigmoid(theta - beta)
    plt.plot(theta, prob, label=f'$\\beta = {beta}$', linewidth=2)

plt.axhline(y=0.5, color='gray', linestyle='--', alpha=0.5)
plt.xlabel('Ability ($\\theta$)', fontsize=12)
plt.ylabel('$P(Y = 1)$', fontsize=12)
plt.legend(title='Item Difficulty')
plt.grid(True, alpha=0.3)
plt.show()
```

2.2.1.2 The Two-Parameter Logistic Model (2PL)

The Rasch model assumes all items have the same *discrimination*—the same slope of the ICC. The **two-parameter logistic model** relaxes this assumption:

DEFINITION: 2PL MODEL

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

where $a_j > 0$ is the discrimination parameter for item j.

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.

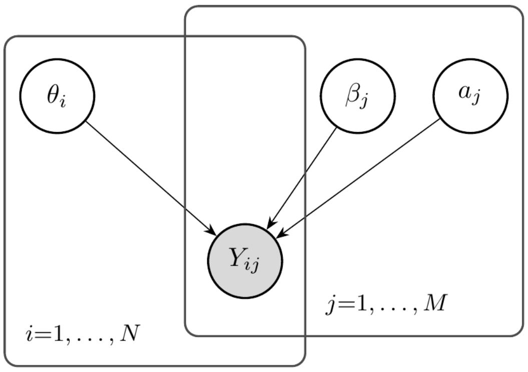

Figure 2.2
Plate diagram for the 2PL model. Each item now has both a difficulty β_j and a discrimination a_j parameter.

```python
#| label: icc-2pl
#| autorun: true
#| fig-cap: "Item Characteristic Curves for 2PL model items with different
  ↪  discriminations. Higher discrimination (steeper slope) means the item better
  ↪  distinguishes between ability levels."

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

# Items with same difficulty but different discriminations
beta = 0
discriminations = [0.5, 1.0, 1.5, 2.0]

plt.figure()
for a in discriminations:
    prob = sigmoid(a * (theta - beta))
    plt.plot(theta, prob, label=f'$a = {a}$', linewidth=2)

plt.axhline(y=0.5, color='gray', linestyle='--', alpha=0.5)
plt.axvline(x=0, color='gray', linestyle='--', alpha=0.5)
plt.xlabel('Ability ($\\theta$)', fontsize=12)
plt.ylabel('$P(Y = 1)$', fontsize=12)
plt.title('2PL Model: Effect of Discrimination ($\\beta = 0$)', fontsize=14)
plt.legend(title='Discrimination')
plt.grid(True, alpha=0.3)
plt.show()
```

2.2.1.3 *The Three-Parameter Logistic Model (3PL)*

For multiple-choice tests, even low-ability test-takers may answer correctly by guessing. The **three-parameter logistic model** adds a lower asymptote:

Definition: 3PL Model

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

where $c_j \in [0, 1]$ is the guessing (or pseudo-chance) parameter.

For a 4-option multiple-choice item, we might expect $c_j \approx 0.25$ if low-ability test-takers guess randomly.

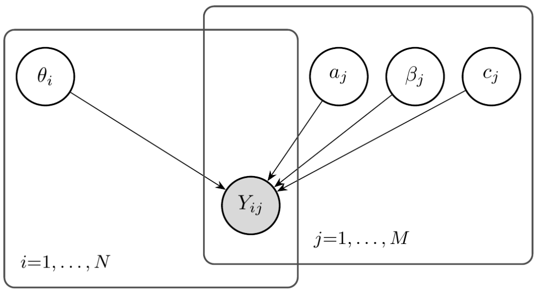

Figure 2.3
Plate diagram for the 3PL model. The guessing parameter c_j sets a lower asymptote on the response probability.

```
#| label: icc-3pl
#| autorun: true
#| fig-cap: "Comparison of 1PL, 2PL, and 3PL models. The 3PL has a non-zero lower
↪  asymptote representing guessing."

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

# Compare the three models
a, beta, c = 1.5, 0, 0.25
```

```python

p_1pl = sigmoid(theta - beta)
p_2pl = sigmoid(a * (theta - beta))
p_3pl = c + (1 - c) * sigmoid(a * (theta - beta))

plt.figure()
plt.plot(theta, p_1pl, label='1PL (Rasch)', linewidth=2)
plt.plot(theta, p_2pl, label='2PL', linewidth=2)
plt.plot(theta, p_3pl, label='3PL', linewidth=2)

plt.axhline(y=0.5, color='gray', linestyle='--', alpha=0.3)
plt.axhline(y=c, color='gray', linestyle=':', alpha=0.5, label=f'Guessing = {c}')
plt.xlabel('Ability ($\\theta$)', fontsize=12)
plt.ylabel('$P(Y = 1)$', fontsize=12)
plt.title('Comparison of IRT Models', fontsize=14)
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

2.2.2 *Factor Models*

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.

2.2.2.1 *The Linear Factor Model*

The classical linear factor model assumes observed variables are linear combinations of latent factors plus noise:

Definition: Linear Factor Model

$$X_j = \lambda_{j1}F_1 + \lambda_{j2}F_2 + \cdots + \lambda_{jK}F_K + \epsilon_j$$

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.

2.2.2.2 *The Logistic Factor Model*

For binary data, we use a logistic link function:

Definition: Logistic Factor Model

$$P(Y_{ij} = 1 | U_i, V_j, Z_j) = \sigma(U_i^\top V_j + Z_j)$$

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
- $Z_j \in \mathbb{R}$ is the item intercept

This is the model used in later chapters of AIMS for multidimensional AI evaluation.

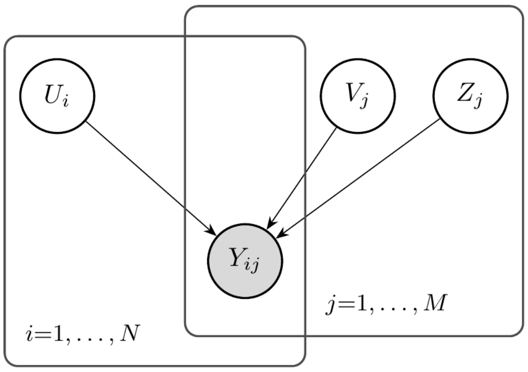

Figure 2.4
Plate diagram for the logistic factor model. The latent factor vector U_i interacts with item-specific loadings V_j and intercepts Z_j to produce responses.

2.2.2.3 *Connection Between IRT and Factor Analysis*

The Rasch model is equivalent to a one-factor logistic model with equal loadings:

Theorem: Rasch-Factor Equivalence

The Rasch model $P(Y_{ij} = 1) = \sigma(\theta_i - \beta_j)$ is equivalent to a single-factor logistic model with $U_i = \theta_i$, $V_j = 1$ for all j, and $Z_j = -\beta_j$.

More generally, multidimensional IRT models and logistic factor models are closely related, differing primarily in parameterization and estimation approach.

2.2.2.4 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:

$$S_{jk} = \mathrm{Cor}(Y^*_{ij}, U_{ik}) = \frac{(V_j^\top \Sigma)_k}{\sqrt{\Sigma_{kk}}\sqrt{V_j^\top \Sigma V_j + \pi^2/3}}$$

where $\Sigma = \mathrm{Cov}(U)$ is the factor covariance matrix.

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.

2.2.2.5 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.

2.2.3 Paired Comparison Models: Elo and Bradley-Terry

Not all measurement data comes in the form of item responses. In many settings, we observe *pairwise comparisons*: which of two items is preferred, which of two players wins. These settings require different models.

2.2.3.1 The Bradley-Terry Model

The Bradley-Terry model (1952) is the foundational model for paired comparisons:

Definition: Bradley-Terry Model

$$P(\text{item } i \text{ beats item } j) = \frac{\exp(\theta_i)}{\exp(\theta_i) + \exp(\theta_j)} = \sigma(\theta_i - \theta_j)$$

where θ_i is the "strength" or "quality" of item i.

The model has the same mathematical form as the Rasch model, but the interpretation differs: instead of a person answering an item, we have two items competing against each other.

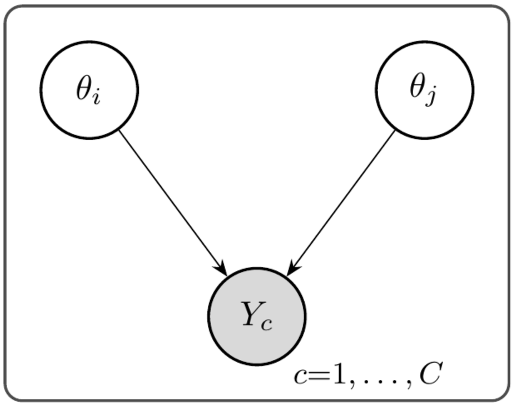

Figure 2.5
Plate diagram for the Bradley-Terry model. A comparison outcome Y_c depends on the strengths θ_i and θ_j of the two competitors in each pair.

2.2.3.2 The Elo Rating System

The Elo rating system, developed by Arpad Elo for chess ratings, is essentially a Bradley-Terry model with online updates:

DEFINITION: ELO RATING SYSTEM

After player i with rating R_i plays player j with rating R_j, the ratings are updated:

$$R_i^{\text{new}} = R_i + K(S_i - E_i)$$

where:

- $S_i \in \{0, 0.5, 1\}$ is the actual outcome (loss, draw, win)
- $E_i = \sigma((R_j - R_i)/400 \cdot \ln 10)$ is the expected outcome
- 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.

2.2.3.3 Connection to AI Evaluation

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.

Chatbot Arena as Thurstone's Comparative Judgment

The Chatbot Arena implements exactly the paradigm that L.L. Thurstone proposed in 1927: measuring psychological attributes through pairwise comparisons. Thurstone developed this method to scale attitudes, preferences, and other subjective quantities. A century later, the same mathematics underlies how we rank AI systems.

2.2.4 Network Models: GGM and Ising

The models discussed so far assume a *common cause* structure: latent variables cause observed responses. Network models propose an alternative: observed variables are directly connected to each other, and correlations arise from these connections rather than common latent causes.

2.2.4.1 The Gaussian Graphical Model (GGM)

For continuous data, the Gaussian Graphical Model represents conditional independence relationships:

Definition: Gaussian Graphical Model

Variables $X = (X_1, \dots, X_M)$ follow a multivariate normal distribution with precision matrix $\Omega = \Sigma^{-1}$. The partial correlation between X_j and X_k given all other variables is:

$$\rho_{jk\cdot\text{rest}} = -\frac{\Omega_{jk}}{\sqrt{\Omega_{jj}\Omega_{kk}}}$$

Two variables are conditionally independent if and only if $\Omega_{jk} = 0$.

The GGM can be visualized as a network where nodes are variables and edges represent non-zero partial correlations.

2.2.4.2 The Ising Model

For binary data, the Ising model (borrowed from statistical physics) provides an analogous framework:

Definition: Ising Model

$$P(Y = y) = \frac{1}{Z} \exp\left(\sum_j \tau_j y_j + \sum_{j<k} \omega_{jk} y_j y_k\right)$$

where:

- τ_j are threshold parameters (similar to item difficulties)
- ω_{jk} are interaction parameters (edge weights)
- Z is a normalizing constant

In the Ising model, the correlation between two items arises from their direct connection ω_{jk}, not from a common latent factor.

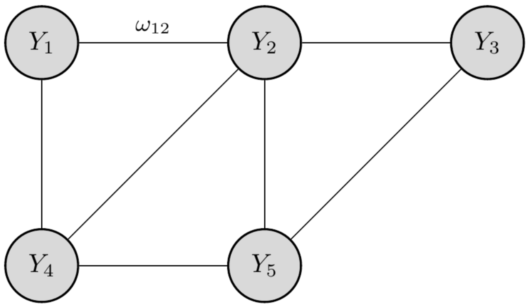

Figure 2.6
Graphical model for the Ising model. Unlike latent variable models, all nodes are observed and correlations arise from direct pairwise connections ω_{jk}.

2.2.4.3 Network vs. Latent Variable Models

The choice between network and latent variable models reflects different theories about what causes observed correlations:

Aspect	Latent Variable Model	Network Model
Cause of correlations	Common latent factor	Direct connections
Removing an item	No effect on other correlations	May reduce correlations
Theoretical commitment	Constructs exist and cause responses	Constructs are summaries
Example	"Intelligence" causes good performance	Skills directly cause each other

For AI evaluation, the question is whether benchmark items are indicators of a common capability (latent variable view) or whether they form a network of related but distinct skills (network view). This distinction has implications for how we aggregate performance across items.

Which Model for AI?

The choice between latent variable and network models is not merely technical—it reflects different beliefs about AI capabilities:

– **Latent variable view**: Models have underlying capabilities (reasoning, knowledge, language understanding) that cause their benchmark performance.
– **Network view**: Benchmark items measure distinct skills that may reinforce each other but do not share a common cause.

Both views may be partially correct. AIMS primarily adopts the latent variable view but acknowledges that some benchmark items may not fit this framework.

2.2.5 Hierarchical Models

The models introduced so far treat all items as exchangeable—a single set of item parameters enters the likelihood without any grouping structure. In practice, AI evaluations are *nested*: items belong to benchmarks, benchmarks belong to suites or domains, and suites may be grouped into broader capability areas. Hierarchical (multilevel) models make this nesting explicit in the model specification, treating it as part of the data-generating process rather than an afterthought of analysis.

Definition: Hierarchical IRT Model

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

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

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

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

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.

The decision to include hierarchical structure is a *modeling* choice, analogous to the decision between the 1PL and 2PL. It encodes the assumption that items within the same benchmark share difficulty characteristics—their parameters are not independent draws from a single global distribution, but cluster by benchmark. Ignoring this structure and treating all items as exchange-

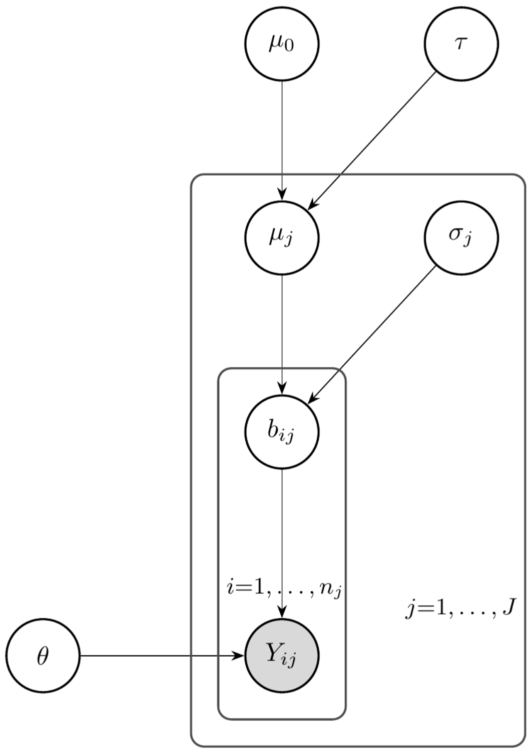

Figure 2.7
Plate diagram for the hierarchical IRT model. Item difficulties b_{ij} are drawn from benchmark-level distributions parameterized by μ_j and σ_j, which may themselves be drawn from domain-level hyperparameters μ_0 and τ. Nested plates reflect the hierarchical data structure.

able conflates within-benchmark and between-benchmark variation, producing estimates that may not generalize beyond the specific 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

Explicitly modeling these levels separates the sources of variation at each level. This enables principled generalization from the benchmarks actually tested to the broader construct they are intended to measure. Estimation methods for hierarchical models, including partial pooling and Bayesian inference, are covered in Chapter 2.

2.3 The Rasch Model as "The Measurement Model"

Among the many probabilistic models for measurement, the Rasch model holds a special status. Georg Rasch and his followers argue that it is not merely *one* measurement model among many—it is *the* measurement model, the only model that satisfies the requirements for fundamental measurement. This section examines this claim carefully.

2.3.1 Sufficiency of Sum Scores

The most remarkable property of the Rasch model is that the **sum score is a sufficient statistic** for the ability parameter. This means that the total number of correct responses contains all the information about a person's ability; knowing *which* items were answered correctly adds nothing.

Theorem: Sufficiency in the Rasch Model

In the Rasch model, the total score $S_i = \sum_{j=1}^{M} Y_{ij}$ is a sufficient statistic for the ability parameter θ_i. That is:

$$P(Y_i|S_i,\theta_i) = P(Y_i|S_i)$$

The conditional distribution of the response pattern given the sum score does not depend on θ.

Proof

The joint probability of response pattern $Y_i = (Y_{i1}, \dots, Y_{iM})$ is:

$$P(Y_i|\theta_i, \beta) = \prod_{j=1}^{M} \frac{\exp(Y_{ij}(\theta_i - \beta_j))}{1 + \exp(\theta_i - \beta_j)}$$

Expanding:

$$= \frac{\exp(\theta_i \sum_j Y_{ij}) \cdot \exp(-\sum_j Y_{ij}\beta_j)}{\prod_j (1 + \exp(\theta_i - \beta_j))}$$

$$= \frac{\exp(\theta_i S_i) \cdot \exp(-\sum_j Y_{ij}\beta_j)}{\prod_j (1 + \exp(\theta_i - \beta_j))}$$

The likelihood factors as $L(\theta|Y) = g(S, \theta) \cdot h(Y, \beta)$, where g depends on θ only through S.

By the factorization theorem, S is sufficient for θ.

To see that the conditional distribution $P(Y_i|S_i)$ does not depend on θ:

$$P(Y_i|S_i, \theta_i) = \frac{P(Y_i|\theta_i, \beta)}{P(S_i|\theta_i, \beta)}$$

Both numerator and denominator contain the factor $\exp(\theta_i S_i)$, which cancels when S_i is fixed.

2.3.1.1 *Why Sufficiency Matters*

Sufficiency has profound implications:

1. **Data reduction without information loss.** We can summarize each person's responses by a single number (the sum score) without losing any information about their ability.
2. **Justification for sum scores.** The common practice of computing total scores is justified *only if* the Rasch model holds. Under other models, sum scores discard information.
3. **Conditional inference.** We can estimate item parameters without knowing person parameters, and vice versa, by conditioning on sufficient statistics.

Sufficiency and AI Benchmarks

When we compute a model's accuracy on a benchmark, we are computing a sum score (proportion correct = sum / number of items). This is appropriate if the Rasch model holds. But if items have different discriminations, the sum score loses information—we should weight some items more than others.

Implication: Before trusting aggregate benchmark scores, we should test whether the Rasch model fits the data.

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.

2.3.2 *Specific Objectivity*

Georg Rasch's central contribution was not the mathematical model itself (which had been proposed earlier by others) but the philosophical framework of **specific objectivity**.

Definition: Specific Objectivity

A measurement procedure exhibits **specific objectivity** if comparisons between persons are independent of which items are used:

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

The item parameter β_j cancels completely. Similarly, comparisons between items are independent of which persons are used.

In the Rasch model, the odds ratio for two persons on the same item is:

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

The item difficulty β_j cancels! This means person comparisons are the same regardless of which item we use.

Rasch identified two levels of objectivity:

1. **Local objectivity**: Comparisons are item-independent for a specific pair of persons.
2. **General objectivity**: The entire ability scale is sample-independent. Ability estimates remain valid regardless of which items were administered.

2.3.3 *Test-Free and Sample-Free Measurement*

Specific objectivity enables what Rasch called **test-free person measurement** and **sample-free item calibration**:

– **Test-free person measurement**: A person's ability can be estimated from *any* subset of calibrated items, and the estimate will be the same (within sampling error).

– **Sample-free item calibration**: An item's difficulty can be estimated from *any* sample of persons, and the estimate will be the same.

This is remarkable because it mirrors the properties of physical measurement:

The Analogy to Physical Measurement

Consider measuring temperature with different thermometers:

– A mercury thermometer in New York should give the same reading as an alcohol thermometer in London for the same temperature.
– Calibrating a thermometer on hot water should yield parameters that work equally well for cold water.

The Rasch model claims the same properties for psychological measurement: calibrated tests yield the same ability estimates regardless of which specific items are used.

2.3.3.1 *Implications for AI Evaluation*

If AI benchmarks satisfy Rasch model assumptions:

1. **Benchmark subset comparisons are valid**. We can compare a model tested on MMLU subset A with a model tested on MMLU subset B, as long as both subsets are calibrated to the same scale.
2. **New questions can be calibrated on any models**. We can add new questions to a benchmark by testing them on a sample of models, then use them to evaluate future models.
3. **Adaptive testing becomes possible**. We can select questions dynamically based on a model's performance, arriving at an accurate ability estimate with fewer questions.
4. **Cross-benchmark comparisons may be possible**. If different benchmarks measure the same construct, we can equate their scales.

These properties are not guaranteed—they hold only if the Rasch model fits the data. Testing model fit becomes essential.

2.3.4 The Rasch vs. General IRT Debate

The claim that Rasch is "the" measurement model is controversial. The debate centers on the **prescriptive vs. descriptive** approaches to measurement.

2.3.4.1 The Prescriptive Approach (Rasch School)

The Rasch school argues:

1. **Measurement requires specific objectivity**. Without it, we cannot make person comparisons that are independent of the test used.
2. **The model is a requirement, not a description**. If data do not fit the Rasch model, the items do not measure the same construct. We should discard misfitting items, not adopt a more complex model.
3. **Discrimination variation is a problem, not a feature**. Items with different discriminations measure the construct with different precision. Mixing them produces a heterogeneous test that does not measure a single thing.
4. **Sufficiency is non-negotiable**. The sum score must be sufficient for ability, or we are not measuring anything meaningful.

2.3.4.2 The Descriptive Approach (General IRT)

The general IRT school responds:

1. **Models should fit data**. The purpose of a statistical model is to describe the data accurately. If items have different discriminations, we should model this, not ignore it.
2. **Perfect fit is unrealistic**. Real data never perfectly fit any model. The Rasch school's insistence on exact fit is impractical.
3. **Information is lost by forcing Rasch**. Discarding items that don't fit Rasch means discarding information. Better to use all items and model their characteristics.
4. **2PL/3PL models are more realistic**. Most tests have items with varying discrimination and guessing. Pretending otherwise does not make it true.

The Fundamental Tension

Prescriptive view: The Rasch model defines what measurement IS. Items that don't fit should be discarded because they don't measure the same thing.

Descriptive view: Use whatever model fits the data best. The 2PL/3PL models are more realistic for most applications.

This is not merely a statistical disagreement—it reflects different philosophies of science. The Rasch school treats measurement theory as providing *requirements* that data must satisfy. The IRT school treats models as *tools* that should be chosen based on fit.

2.3.4.3 *Implications for AI Evaluation*

This debate has direct implications for AI benchmark design:

Question	Rasch School Answer	General IRT Answer
Should we allow items with different discriminations?	No—they measure different constructs	Yes—model the discrimination
What if data don't fit Rasch?	Remove misfitting items	Use 2PL or 3PL
Is sum score the right metric?	Yes, if Rasch fits	Only approximately
Can we compare models across benchmarks?	Yes, with Rasch	Requires complex equating

AIMS takes a pragmatic position: we test whether data fit Rasch-like models, use the simpler model when it fits adequately, and acknowledge when more complex models are needed. The key insight is that the choice has *consequences* for what we can conclude from evaluation results.

2.4 *Historical Development*

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.

2.4.1 *Thurstone's Law of Comparative Judgment (1927)*

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'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.

Thurstone's Model (Case V)

Each stimulus i has a true scale value θ_i. When comparing stimuli i and j, each is perceived with Gaussian noise:

$$\tilde{\theta}_i \sim N(\theta_i, \sigma^2), \quad \tilde{\theta}_j \sim N(\theta_j, \sigma^2)$$

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 was revolutionary: it showed that subjective quantities could be measured scientifically. The same mathematics now underlies how we rank AI systems from human preferences.

2.4.2 *Bradley-Terry and Luce (1952-1959)*

In 1952, Ralph Bradley and Milton Terry developed a model for ranking from paired comparisons in the context of incomplete block designs. 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 his **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.

2.4.3 Georg Rasch and the Danish School (1960)

Georg Rasch was a Danish mathematician who worked on problems in educational testing. In 1960, he published "Probabilistic Models for Some Intelligence and Attainment Tests," which introduced what we now call the Rasch model.

Rasch's contribution was not the mathematical model itself—the same formula had appeared earlier in other contexts. His 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*.

Other important figures in the Rasch tradition:

- **Erling Andersen** (Copenhagen): Developed the theory of conditional maximum likelihood estimation for Rasch models.
- **Gerhard Fischer** (Vienna): Extended the Rasch model to the Linear Logistic Test Model (LLTM) and developed software for estimation.
- **David Andrich** (Australia): Extended Rasch models to polytomous data (rating scales) and developed the RUMM software.

2.4.4 McFadden and Econometrics (1974)

In 1974, Daniel McFadden (who later won the Nobel Prize in Economics) developed the random utility framework for discrete choice. His insight was that choices could be modeled as utility maximization with random error:

A person chooses alternative i over j if $U_i + \epsilon_i > U_j + \epsilon_j$, where U is deterministic utility and ϵ is random. If the errors are i.i.d. Gumbel distributed, this yields the logistic choice model.

McFadden's work connected preference models to economics and provided a theoretical justification for the Bradley-Terry model: it arises from random utility maximization under specific distributional assumptions.

2.4.5 Modern Developments

Network Psychometrics (2010s): Borsboom and colleagues proposed that psychological constructs might be better understood as networks of causally connected symptoms rather than reflections of underlying latent variables. The Ising model and Gaussian Graphical Model provide statistical tools for this perspective.

AI Evaluation (2020s): The application of psychometric methods to AI evaluation is recent. Key developments include:

- Chatbot Arena using Elo ratings for LLM ranking (LMSYS, 2023)

– Application of IRT to benchmark analysis (Polo et al., 2024)
– Multidimensional factor models for AI capabilities (this textbook)

Key Historical Papers

Foundations:

– Thurstone, L.L. (1927). A law of comparative judgment. *Psychological Review*, 34, 273–286.
– Bradley, R.A. & Terry, M.E. (1952). Rank analysis of incomplete block designs. *Biometrika*, 39, 324–345.
– Luce, R.D. (1959). *Individual Choice Behavior: A Theoretical Analysis*. Wiley.
– Rasch, G. (1960). *Probabilistic Models for Some Intelligence and Attainment Tests*. Danish Institute for Educational Research.
– McFadden, D. (1974). Conditional logit analysis of qualitative choice behavior. In *Frontiers in Econometrics* (pp. 105–142). Academic Press.

Modern:

– Borsboom, D. (2005). *Measuring the Mind: Conceptual Issues in Contemporary Psychometrics*. Cambridge University Press.
– Epskamp, S. et al. (2018). The Gaussian graphical model in cross-sectional and time-series data. *Multivariate Behavioral Research*, 53, 453–480.

2.5 From Psychology to AI: Transferring Measurement Science

The concepts developed in psychology and education transfer directly to AI evaluation. This section makes the mapping explicit and highlights both the parallels and the differences.

2.5.1 The Translation Table

Psychology/Education	AI Evaluation	Symbol
Test taker (person, examinee)	AI model	i
Test item (question, problem)	Benchmark question	j
Ability, trait, latent construct	Capability, skill	θ_i or U_i
Item difficulty	Question difficulty	β_j or V_j
Item discrimination	Question informativeness	a_j
Response (correct/incorrect)	Model output (correct/incorrect)	Y_{ij}
Test (collection of items)	Benchmark (collection of questions)	–
Sum score (number correct)	Accuracy	S_i
Reliability	Evaluation consistency	–
Validity	Measuring intended capability	–
Test bias (DIF)	Benchmark contamination/bias	–

Psychology/Education	AI Evaluation	Symbol
Adaptive testing (CAT)	Efficient evaluation	–

2.5.2 Key Parallels

Reliability. In educational testing, reliability refers to the consistency of scores across different conditions:

- *Test-retest reliability:* Does the same person get the same score on repeated testing?
- *Internal consistency:* Do items within the test correlate with each other?
- *Standard error of measurement:* How precise is the score estimate?

For AI evaluation:

- *Run-to-run consistency:* Does the same model get the same score with different random seeds?
- *Item consistency:* Do benchmark questions correlate with each other?
- *Confidence intervals:* How uncertain is the accuracy estimate?

Validity. In educational testing, validity concerns whether the test measures what it claims to measure:

- *Content validity:* Do the items adequately sample the domain?
- *Criterion validity:* Does the score predict relevant outcomes?
- *Construct validity:* Does the score behave as theory predicts?

For AI evaluation:

- *Content validity:* Does the benchmark cover the intended capability domain?
- *Criterion validity:* Does benchmark performance predict real-world usefulness?
- *Construct validity:* Do models that score high actually have the intended capability?

Fairness and Bias. In educational testing, differential item functioning (DIF) analysis checks whether items are biased against certain groups:

- An item shows DIF if persons with equal ability but different group membership have different probabilities of answering correctly.

For AI evaluation:

- *Training data contamination:* Did some models see the test questions during training?
- *Architecture bias:* Are some questions easier for certain model architectures?
- *Prompt sensitivity:* Do different prompt formats advantage different models?

2.5.3 Key Differences

While the mathematical framework transfers directly, some differences are worth noting:

1. **Number of items**. Psychological tests typically have tens to hundreds of items. AI benchmarks may have thousands or hundreds of thousands. This affects estimation and model fitting.
2. **Deterministic responses**. Human test-takers show stochastic variation—they may answer the same question differently on different occasions. AI models (with temperature 0) are often deterministic. This changes how we interpret probability models.
3. **Construct definition**. Psychological constructs like "intelligence" or "anxiety" have extensive theoretical literature. AI capabilities like "reasoning" or "common sense" are less well defined.
4. **Speed of change**. Human abilities change slowly. AI capabilities can change dramatically with each model release. This affects the stability of calibrations.
5. **Population structure**. Human populations have known demographic structures. The "population" of AI models is arbitrary—determined by which models researchers choose to evaluate.

2.5.4 Case Study: The Chatbot Arena

The Chatbot Arena (LMSYS) provides a concrete example of measurement concepts applied to AI:

Setting: Users interact with two anonymous language models and vote for the one they prefer. Models are ranked using Elo ratings computed from these pairwise comparisons.

Measurement framework: This is exactly Thurstone's comparative judgment paradigm from 1927. The Elo rating system implements Bradley-Terry maximum likelihood estimation with online updates.

Validity questions:

– What construct do the ratings measure? "Human preference" is vague. Preferences for what—helpfulness, harmlessness, style, factual accuracy?
– Are ratings stable across different user populations?
– Do ratings predict performance on other benchmarks or real-world tasks?

Reliability questions:

– How many comparisons are needed for stable ratings?
– How sensitive are ratings to the specific prompts used?
– Do ratings fluctuate as new models enter the arena?

The Arena demonstrates both the power and limitations of measurement approaches. The Elo ratings provide a principled summary of human preferences, but interpreting what they mean requires the full apparatus of validity theory.

2.6 Cold-Start Prediction

The factor models introduced in Section 2.2.2 can predict held-out entries in the response matrix when trained with appropriate masking schemes (see Section 3.7 in Chapter 3). But what if we want to evaluate a *brand-new* model or benchmark item that has never been tested? This is the **cold-start problem**: predicting correctness without any observed responses for the target model or item.

Prediction-Powered Evaluation (PPE) addresses this by learning mappings from external features (question embeddings, model metadata) to the latent parameters of a factor model, enabling zero-shot correctness prediction. PPE is an *amortized* version of the factor model: instead of optimizing latent parameters per entity, it learns a function that maps observable features directly to the latent space.

2.6.1 The Semantic–Behavioral Gap

A natural idea is to predict behavioral similarity from semantic similarity: if two questions have similar embeddings, perhaps models respond to them similarly. However, this intuition is misleading.

We compare:

- **Semantic similarity**: cosine similarity between question embeddings
- **Behavioral similarity**: tetrachoric correlation between model responses

$$\text{Corr}_{\text{semantic}}(i,j) = \cos(E_i, E_j), \quad \text{Corr}_{\text{behavioral}}(i,j) = \text{TetraCorr}(Y_{\cdot i}, Y_{\cdot j})$$

Even near-identical embeddings (cosine > 0.99) exhibit nearly random behavioral correlations (−1 to +1), showing that semantic proximity does *not* imply behavioral equivalence. This motivates a more structured approach.

2.6.2 The PPE Pipeline

The prediction-powered framework has three stages:

Stage 1 — Factor Model Pretraining. We first learn latent behavioral factors (U, V, Z) from observed response data Y_{ij} using the factor model from Section 2.2.2:

$$p(Y_{ij} = 1 \mid U_i, V_j, Z_j) = \sigma(U_i^\top V_j + Z_j)$$

Stage 2 — Prediction-Powered Mapping. Two parallel predictors are trained:

- **Item-side predictor** f_V: maps question embeddings $E_j \in \mathbb{R}^{4096}$ to latent parameters $[\hat{V}_j, \hat{Z}_j] = f_\theta(E_j)$
- **Model-side predictor** f_U: maps model metadata $F_i \in \mathbb{R}^{24}$ (scale, architecture, release time) to $\hat{U}_i = F_i W_U$

The item predictor is a neural network trained with Bernoulli log-likelihood using fixed U from Stage 1. The model predictor is a simple linear transformation for interpretability.

Stage 3 — Cold-Start Evaluation. Given a new model or item, we predict its latent parameters from metadata/embeddings and reconstruct correctness probabilities:

$$\hat{P}_{ij} = \sigma(\hat{U}_i^\top \hat{V}_j + \hat{Z}_j)$$

Component	Input	Output	Purpose
Factor model	Response matrix Y	U, V, Z	Extract latent behavioral structure
Semantic predictor	Question embeddings E_j	$[\hat{V}_j, \hat{Z}_j]$	Generalize to unseen questions
Model predictor	Metadata F_i	$\hat{U}_i$	Generalize to unseen models
Correctness predictor	$\hat{U}_i, \hat{V}_j, \hat{Z}_j$	$\hat{P}_{ij}$	Predict correctness without running evaluation

2.6.3 *Iterative Filtering*

Before training the PPE pipeline, we can improve inter-item consistency by removing adversarial or inconsistent items. **Iterative filtering** repeatedly computes tetrachoric correlations and removes items with the highest proportion of negative correlations. After filtering, negative correlations typically drop from ~23% to under 2%, retaining roughly half the original items.

2.6.4 *Empirical Results*

Typical results on held-out data:

Split	AUC
randcol–randcol	0.804
randrow–randrow	0.848

The strong AUC on row holdout (unseen models) confirms that model behavior is well-predicted by simple metadata features, enabling reliable evaluation without running models on every benchmark item.

2.7 Summary and Preview

This chapter has introduced the measurement science framework that underlies the rest of AIMS. The key ideas are:

2.7.1 Key Takeaways

1. **Measurement requires more than scoring**. Assigning numbers to models based on benchmark performance is scoring, not measuring. Measurement requires a theory connecting scores to latent constructs.
2. **Validity matters**. A benchmark measures a construct only if the construct exists and causally produces variation in scores. The philosophical and diagnostic foundations of validity are developed in Chapter 6.
3. **The Rasch model has special properties**. Sufficiency of sum scores and specific objectivity make Rasch uniquely suitable for fundamental measurement. These properties justify treating sum scores as measurements.
4. **Multiple models exist for different purposes**. IRT models (1PL, 2PL, 3PL), factor models, paired comparison models (Bradley-Terry, Elo), network models (GGM, Ising), and hierarchical models serve different purposes. The choice of model—including whether to represent nested evaluation structure—has implications for what we can conclude.
5. **Psychology solved these problems decades ago**. The tools developed in psychometrics—reliability, validity, dimensionality analysis, adaptive testing—apply directly to AI evaluation.

2.7.2 Preview of Following Chapters

The chapters that follow apply this framework to specific AI evaluation challenges:

– **Chapter 2 (Learning)**: Covers parameter estimation for IRT and factor models, including maximum likelihood, EM algorithms, Bayesian inference, regularization, and model selection. It also develops generalization experiments and the cold-start prediction problem — predicting performance for unseen models and items.

– **Chapter 3 (Design)**: Applies measurement principles to benchmark design, addressing how to construct valid and reliable AI evaluations.

– **Chapter 4 (Reliability)**: Develops the theory of measurement precision, including classical and IRT-based reliability, standard errors, and information functions.

The measurement concepts from this chapter recur throughout. When we justify using sum scores, we appeal to sufficiency in the Rasch sense. When we analyze benchmark dimensionality, we apply the factor models introduced in this chapter and trained using methods from Chapter 2. The question of whether benchmarks measure what they claim to measure — validity — is developed in Chapter 6.

2.8 *Exercises*

2.8.1 *Theoretical Exercises*

Exercise 1.1 (*): Explain in your own words 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.

2.8.2 *Computational Exercises*

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

```python
# 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
    # YOUR 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.

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

```python
#
# 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)
    """
    # YOUR 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.

```python
# 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
    """
    # YOUR CODE HERE
    pass
```

2.8.3 Discussion Questions

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: Sang Truong 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?

2.9 Bibliographic Notes

2.9.1 Item Response Theory

The standard reference for IRT is Lord and Novick's *Statistical Theories of Mental Test Scores* (1968), though it predates modern computational methods. More accessible introductions include Hambleton and Swaminathan's *Item Response Theory* (1985) and de Ayala's *The Theory and Practice of Item Response Theory* (2009). The *Handbook of Modern Item Response Theory* (van der Linden & Hambleton, 1997) provides comprehensive coverage.

2.9.2 Rasch Measurement

Rasch's original book *Probabilistic Models for Some Intelligence and Attainment Tests* (1960) remains influential. Wright and Stone's *Best Test Design* (1979) provides practical guidance. Fischer and Molenaar's *Rasch Models: Foundations, Recent Developments, and Applications* (1995) covers extensions and applications. For the philosophical foundations, see Rasch's papers on objectivity collected in the *Rasch Measurement Transactions* archive.

2.9.3 Historical Development

Thurstone's seminal paper "A Law of Comparative Judgment" (1927) launched the quantitative study of preferences. Bradley and Terry's "Rank Analysis of Incomplete Block Designs" (1952) and Luce's *Individual Choice Behavior* (1959) established the axiomatic foundations. McFadden's "Conditional Logit Analysis" (1974) connected these to economic theory. For a history of psychometrics, see Boring's *A History of Experimental Psychology* (1950).

2.9.4 Network Psychometrics

The network approach is developed in Borsboom and Cramer's "Network Analysis: An Integrative Approach" (2013) and formalized in Epskamp et al.'s papers on the Gaussian graphical model and Ising model (2018). The *Network Psychometrics with R* book (Epskamp et al., 2022) provides practical guidance.

2.9.5 AI Evaluation

The application of psychometric methods to AI is recent. For IRT applied to LLMs, see Polo et al.'s "Efficient Multi-Prompt Evaluation" (2024). For factor models, see the methods developed in this textbook. The Chatbot Arena is described in Zheng et al.'s "Judging LLM-as-a-Judge" (2023).

2.9.6 Scaling Laws and IRT

Sang Truong et al. (2025) introduce Item Response Scaling Laws (IRSL), which embed IRT within the scaling law framework. By factorizing model ability from question difficulty, IRSL reduces scaling law estimation from $O(M \times N)$ to $O(M + N)$ parameters, with the estimated ability transferring across benchmarks. Their Beta-IRT formulation models empirical probability responses (token probabilities, pass rates) rather than binary correctness, using a Beta loss that achieves reliable calibration with as few as 2 test takers. Schaeffer et al. (2025) provide the distributional theory explaining why per-problem exponential scaling aggregates to power-law scaling: the exponent is determined by the left-tail shape parameter of the success probability distribution, connecting the heterogeneity of item difficulty to aggregate scaling behavior.

3 Learning

Intended Learning Outcomes

By the end of this chapter, you 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 descent 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.
10. **Evaluate** the generalization of learned factor models under various masking schemes (entry-wise, row hold-out, column holdout).

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 and Generalization

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

– Generalization experiments and masking schemes (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 **?@sec-notation** for the complete notation reference.

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

3.1 *Why Learning Matters for AI Measurement*

Chapter 1 introduced the measurement models—Rasch, 2PL, factor models—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.

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 Chapter 3 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. We also introduce generalization experiments that evaluate how well learned models transfer to unseen data. The design of *which* items to include in a benchmark, including active item selection, is the subject of Chapter 3.

3.2 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.

3.2.1 The Likelihood Function

Recall from Chapter 1 that the Rasch model specifies the probability of a correct response as:

$$P(Y_{ij} = 1 \mid \theta_i, \beta_j) = \sigma(\theta_i - \beta_j) = \frac{1}{1 + e^{-(\theta_i - \beta_j)}} \tag{3.1}$$

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

Under the assumption of *local independence*—that responses are conditionally independent given the latent parameters—the likelihood of the entire response matrix is:

$$L(\theta, \beta \mid Y) = \prod_{i=1}^{N} \prod_{j=1}^{M} P(Y_{ij} \mid \theta_i, \beta_j)^{Y_{ij}} [1 - P(Y_{ij} \mid \theta_i, \beta_j)]^{1 - Y_{ij}} \tag{3.2}$$

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

$$\ell(\theta, \beta) = \sum_{i=1}^{N} \sum_{j=1}^{M} \left[Y_{ij}(\theta_i - \beta_j) - \log(1 + e^{\theta_i - \beta_j}) \right] \tag{3.3}$$

This is the objective function we want to maximize.

3.2.2 Gradient Derivation

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

$$\frac{\partial \ell}{\partial \theta_i} = \sum_{j=1}^{M} \left[Y_{ij} - \sigma(\theta_i - \beta_j) \right] \tag{3.4}$$

$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i=1}^{N} \left[\sigma(\theta_i - \beta_j) - Y_{ij}\right] \tag{3.5}$$

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.

3.2.3 Implementation with Gradient Descent

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

```python
#| 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,
})
```

```python
#| 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
```

```python

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()
```

Now we implement MLE via gradient descent:

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

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

# 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}")
```

3.2.4 *The Identifiability Problem*

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.

Without addressing identifiability, gradient descent may drift indefinitely. The re-centering step in our implementation ensures parameters remain anchored.

3.2.5 L-BFGS Optimization

While gradient descent is intuitive, quasi-Newton methods like L-BFGS converge much faster by approximating second-order information:

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

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 descent
print(f"\nCorrelation with GD 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}")
```

3.3 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.

3.3.1 Joint MLE (JMLE)

JMLE simultaneously estimates all parameters by maximizing Equation 3.3. While intuitive, JMLE suffers from the *incidental parameter problem*: 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.

This happens because each person parameter θ_i is estimated from only M observations (their responses to M items), and these “incidental” person parameters introduce bias into the item estimates.

For AI benchmarks with many items (typically $M > 100$), this bias is small in practice. But for smaller tests, JMLE can be problematic.

3.3.2 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:

$$P(Y_i \mid S_i, \beta) = \frac{\exp(-\sum_j Y_{ij}\beta_j)}{\gamma_{S_i}(\beta)} \tag{3.6}$$

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.

3.3.3 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:

$$L(\beta) = \prod_{i=1}^{N} \int P(Y_i \mid \theta, \beta)\, p(\theta)\, d\theta \tag{3.7}$$

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.

3.4 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.

3.4.1 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:

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

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.

3.4.2 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')
```

```python
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}")
```

3.4.3 Multidimensional Extension: The Logistic Factor Model

The methods above focused on the Rasch model, which assumes a single latent dimension. For AI benchmarks that measure multiple capabilities, we extend to the **Logistic Factor Model**:

$$P(Y_{ij} = 1 \mid U_i, V_j, Z_j) = \sigma(U_i^\top V_j + Z_j)$$

where:

- $U_i \in \mathbb{R}^K$ is the K-dimensional latent ability vector for model i
- $V_j \in \mathbb{R}^K$ is the factor loading vector for item j
- $Z_j \in \mathbb{R}$ is the item intercept (capturing overall difficulty)

When $K = 1$ and $V_j = 1$ for all j, this reduces to the Rasch model.

3.4.3.1 Implementation

```python
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 $[0, 1]$

3.4.3.2 *Training with LBFGS*

We train the model by minimizing binary cross-entropy loss:

```python
# 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.

3.5 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.

3.5.1 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):

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

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.

3.5.2 Posterior Computation

Bayes' theorem gives us the posterior distribution:

$$p(\theta, \beta \mid Y) \propto p(Y \mid \theta, \beta) \cdot p(\theta) \cdot p(\beta) \tag{3.8}$$

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

3.5.3 MAP Estimation

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

$$\hat{\theta}_{\text{MAP}}, \hat{\beta}_{\text{MAP}} = \arg\max_{\theta,\beta} \left[\ell(\theta, \beta \mid Y) + \log p(\theta) + \log p(\beta)\right] \tag{3.9}$$

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

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

```python
#| 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.

3.5.4 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
```

```python

    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.

3.6 Regularization and Model Selection

3.6.1 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)

3.6.2 *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.

```python
#| 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

    # Create random fold assignments for entries
    fold_assignment = np.random.randint(0, n_folds, (N, M))

    cv_scores = []
    for fold in range(n_folds):
        train_mask = (fold_assignment != fold).astype(float)
        score = fit_and_evaluate(train_mask, Y, lambda_param)
        cv_scores.append(score)

    return np.mean(cv_scores), np.std(cv_scores)

# Grid search over regularization strengths
lambdas = [0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 5.0]
cv_means = []
cv_stds = []

print("Cross-validation results:")
for lam in lambdas:
    mean, std = cross_validate(Y, lam)
    cv_means.append(mean)
    cv_stds.append(std)
    print(f"  lambda = {lam:5.3f}: CV log-lik = {mean:.4f} +/- {std:.4f}")

# Plot
plt.figure()
plt.errorbar(lambdas, cv_means, yerr=cv_stds, fmt='o-', capsize=5, markersize=8)
plt.xscale('log')
plt.xlabel('Regularization strength ($\\lambda$)')
plt.ylabel('Cross-validation log-likelihood')
plt.title('Hyperparameter Selection via Cross-Validation')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

best_lambda = lambdas[np.argmax(cv_means)]
print(f"\nBest regularization: lambda = {best_lambda}")
```

3.7 Generalization Experiments

The estimation methods developed in this chapter produce learned factor models with latent parameters (U, V, Z). But how well do these models generalize? 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.

3.7.1 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.

3.7.2 Implementation of Masking Functions

```python
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
```

3.7.3 Two-Stage Training for Holdout Generalization

To avoid data contamination in row and column holdout experiments, we use a **two-stage training procedure**:

3.7.3.1 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)
```

3.7.3.2 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.

3.7.4 *Evaluation Across Masking Schemes*

For each masking scheme, we compute AUC on the held-out entries:

```python
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. Sang Truong 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.

3.8 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 3 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?
5. **Multidimensional Extensions**: The chapter focused on unidimensional models (single ability). How would the learning procedures change for multidimensional factor models? What additional challenges arise?

3.9 Bibliographic Notes

3.9.1 Maximum Likelihood Estimation

The theory of maximum likelihood for IRT models is developed comprehensively in Lord and Novick (1968) and (?). The joint MLE approach and its limitations (incidental parameter problem) are discussed in (?). For modern computational approaches, see (?).

3.9.2 Conditional and Marginal MLE

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

3.9.3 EM Algorithm

The general EM algorithm was formalized by (?). Its application to IRT is detailed in (?). For modern treatments, see (?).

3.9.4 Bayesian IRT

Bayesian approaches to IRT were pioneered by (?) and advanced using Gibbs sampling by Algorithm ?? . Modern references include (?) and the software documentation for Stan (?).

3.9.5 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. Sang Truong 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.

3.9.6 Optimization Methods

L-BFGS is described in (?). For deep learning optimizers applied to psychometric models, see (?) for Adam.

3.10 Exercises

3.10.1 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?

3.10.2 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 Sang Truong et al. (2025) that Beta-IRT achieves reliable calibration with far fewer models than Binary-IRT.

3.10.3 Discussion Exercises

Exercise 2.9: Compare the convergence of gradient descent, 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?

Exercise 2.11: Implement a cross-validation procedure for selecting between the Rasch, 2PL, and factor models. Apply it to benchmark data with different numbers of items and models. When does the additional complexity of the 2PL or factor model improve out-of-sample prediction?

4 Efficient Measurement

Intended Learning Outcomes

By the end of this chapter, you 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 **?@sec-notation** for the complete notation reference.

4.1 The Design Problem in AI Evaluation

Chapter 1 introduced the measurement models—Rasch, 2PL, factor models, Bradley-Terry—that formalize how latent abilities generate observed responses. Chapter 2 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 1 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.

4.2 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.

4.2.1 Fisher Information for Item Selection

Recall from Chapter 2 that the Fisher information for item j at ability θ in the Rasch model is:

$$I_j(\theta) = P_j(\theta) \cdot (1 - P_j(\theta)) \tag{4.1}$$

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 $\{j_1, \dots, j_K\}$, 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.

```python
#| 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,
})
```

```python
#| 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, 2, figsize=(6, 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)

# 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

# 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')
```

```python
axes[1].set_title(f'Information Accumulation (  = {theta_test})')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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

4.2.2 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.

4.2.3 CAT Implementation

```python
#| 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

        for j in available_items:
            P_j = sigmoid(theta_hat - beta[j])
            info_j = P_j * (1 - P_j)
            if info_j > best_info:
                best_info = info_j
                best_item = j

        # Administer item (simulate response from true ability)
        P_true = sigmoid(theta_true_i - beta[best_item])
        response = int(np.random.random() < P_true)

        administered.append(best_item)
        responses.append(response)
        available_items.remove(best_item)

        # Update ability estimate via MAP (Newton-Raphson)
        for _ in range(10):
            P_vec = sigmoid(theta_hat - np.array([beta[j] for j 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

        # Posterior variance and reliability
        total_info = np.sum([sigmoid(theta_hat - beta[j]) * (1 - sigmoid(theta_hat -
    ↪  beta[j]))
                             for j in administered])
        posterior_var = 1 / (1/prior_var + total_info)
        reliability = 1 - posterior_var / prior_var

        theta_history.append(theta_hat)
        reliability_history.append(reliability)
        se_history.append(np.sqrt(posterior_var))

        if reliability >= reliability_threshold:
            break

    return {
        'theta_hat': theta_hat, 'n_items': len(administered),
        'reliability_history': reliability_history,
        'theta_history': theta_history, 'se_history': se_history,
        'final_reliability': reliability_history[-1]
    }

def random_selection_simulation(theta_true_i, beta, n_items_max=30,
    ↪  reliability_threshold=0.95):
    """Simulate random item selection for comparison."""
    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")
```

4.2.4 *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 2.6.

4.2.5 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 $\mathrm{tr}(\mathcal{I}^{-1})$—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

```

```python
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()
```

```python
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.

4.2.6 Design for Paired Comparisons

When evaluation is based on paired comparisons (as in the Chatbot Arena from Chapter 1), 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.

4.3 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.

4.4 Bibliographic Notes

4.4.1 Optimal Experimental Design

The theory of optimal experimental design originates with Kiefer (1959). D-optimal design is covered in Atkinson, Donev, and Tobias (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.

4.4.2 Computerized Adaptive Testing

CAT has a rich history beginning with (?). The Fisher information criterion for item selection was developed by (?). For multidimensional CAT, see (?). Applications to AI evaluation are emerging; see (?) for recent work on efficient benchmark design. Sang Truong 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.

4.5 Exercises

4.5.1 Theoretical Exercises

Exercise 3.1 (⋆): 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 (⋆⋆): For the 2PL model $P_j(\theta) = \sigma(a_j(\theta - \beta_j))$, derive the Fisher information and show that it equals $a_j^2 P_j(1 - P_j)$. How does discrimination a_j affect optimal item selection?

Exercise 3.3 (⋆⋆): 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.

4.5.2 Computational Exercises

Exercise 3.4 (⋆⋆): Extend the D-optimal design simulation to a 2-dimensional factor model with $K = 2$. Implement item selection using $j^* = \arg\max_j \det(\mathcal{I} + I_j)$ where $I_j = P_j(1 - P_j)V_j V_j^\top$. Compare the selected item loading vectors to random selection.

Exercise 3.5 (⋆⋆): 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.

II

Measurement Reliability and Validity

5 Reliability

Intended Learning Outcomes

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

1. **Define** reliability in the context of AI evaluation and distinguish it from validity.
2. **Apply** Classical Test Theory to decompose observed scores into signal and noise.
3. **Compute** reliability coefficients (Cronbach's alpha, split-half, test-retest) for AI benchmarks.
4. **Explain** Generalizability Theory and use it to quantify multiple sources of measurement error (items, raters, occasions).
5. **Analyze** sources of noise in AI evaluation: annotation variability, prompt sensitivity, model stochasticity, and benchmark sampling.
6. **Evaluate** the reliability of LLM-as-a-judge evaluation protocols.
7. **Design** evaluation procedures that control for identified sources of unreliability.

Suggested Lecture Plan

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

Lecture 1: Signal and Noise in AI Measurement

- When leaderboards flicker: motivating reliability (15 min)
- Classical Test Theory: true score + error (25 min)
- Reliability coefficients for AI benchmarks (25 min)
- Hands-on: split-half reliability and Cronbach's alpha (10 min)

Lecture 2: Generalizability Theory

- Sources of noise: taxonomy for AI evaluation (15 min)
- From CTT to G-theory: multiple error facets (25 min)
- G-studies and D-studies for benchmark design (25 min)
- Hands-on: variance decomposition and D-study optimization (10 min)

Lecture 3: Reliability in Practice

- LLM-as-a-judge reliability (25 min)
- Inter-rater agreement: kappa and beyond (15 min)
- Reliability under IRT: conditional precision (20 min)
- Design principles for reliable AI evaluation (15 min)

Notation

This chapter introduces Classical Test Theory notation: $X_{ij} = T_i + E_{ij}$ (observed = true + error), $\rho_{XX'}$ (reliability), α (Cronbach's alpha), $\sigma_p^2, \sigma_i^2, \sigma_r^2$ (G-theory variance components), and κ (Cohen's kappa). See **?@sec-notation** for the complete notation reference.

5.1 When Leaderboards Flicker

Imagine you evaluate ten language models on a reasoning benchmark. You report the rankings to your 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 6), 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 6 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 noise that can distort model rankings. This chapter provides the theoretical tools to quantify, decompose, and control these sources of noise, drawing on Classical Test Theory and Generalizability Theory from the measurement sciences, applied throughout to the specific challenges of evaluating AI systems.

5.2 Classical Test Theory for AI Evaluation

5.2.1 The True Score Model

Recall from Chapter 2 the decomposition $X = T + E$: every observed score is the sum of a true score and an error. We now develop this formally.

Let X_{ij} be the observed score for model i on measurement occasion j (where "occasion" might mean a particular run, a particular prompt template, or a particular annotator). The true score T_i is defined as the expectation over hypothetical replications:

$$T_i = \mathbb{E}_j[X_{ij}]$$

The error $E_{ij} = X_{ij} - T_i$ satisfies three axioms:

1. **Zero mean:** $\mathbb{E}_j[E_{ij}] = 0$ for all i
2. **Uncorrelated with true scores:** $\text{Cov}(T_i, E_{ij}) = 0$
3. **Uncorrelated across occasions:** $\text{Cov}(E_{ij}, E_{ik}) = 0$ for $j \neq k$

These assumptions give us the fundamental variance decomposition:

$$\sigma_X^2 = \sigma_T^2 + \sigma_E^2$$

Definition: Reliability

Reliability is the proportion of observed score variance that is true score variance:

$$\rho_{XX'} = \frac{\sigma_T^2}{\sigma_X^2} = \frac{\sigma_T^2}{\sigma_T^2 + \sigma_E^2} = 1 - \frac{\sigma_E^2}{\sigma_X^2}$$

Equivalently, reliability is the squared correlation between observed scores and true scores: $\rho_{XX'} = r_{XT}^2$.

In AI evaluation, the "true score" T_i represents the stable capability of model i on the construct being measured. The "error" E_{ij} encompasses everything that causes the observed score to deviate from this stable capability: prompt variation, sampling randomness, annotator disagreement, item selection, and so on.

This connects directly to the reliability formula used in Chapter 3's CAT stopping rule: $R = 1 - \sigma_{\text{post}}^2/\sigma_{\text{prior}}^2$. The IRT version and CTT version express the same idea—both measure the fraction of variance that is signal rather than noise.

5.2.2 *Standard Error of Measurement*

The standard error of measurement (SEM) translates reliability into a confidence interval around individual scores:

$$\text{SEM} = \sigma_X \sqrt{1 - \rho_{XX'}}$$

For AI benchmarks, SEM has direct practical implications. Suppose a benchmark reports accuracy scores with a standard deviation of $\sigma_X = 5$ percentage points and reliability $\rho_{XX'} = 0.90$. Then:

$$\text{SEM} = 5\sqrt{1 - 0.90} \approx 1.58 \text{ points}$$

A 95% confidence interval around any model's score spans approximately $\pm 2 \times \text{SEM} \approx \pm 3.2$ points. Two models whose scores differ by less than this margin may not be meaningfully different—their ranking could easily reverse on a replication. This is a common situation on crowded leaderboards where dozens of models cluster within a few percentage points of each other.

5.2.3 *Reliability Coefficients*

Since true scores are unobservable, reliability must be estimated from data. Three classical approaches correspond to different experimental designs, each suited to different AI evaluation scenarios.

Test-retest reliability. Administer the same benchmark to the same models on two occasions and correlate the scores. For AI: re-run the evaluation with temperature > 0 (different random seeds), or have the same annotators score the same outputs on two occasions. The Pearson correlation between the two sets of scores estimates $\rho_{XX'}$. This method is simple but conflates all sources of occasion-to-occasion variability into a single number.

Split-half reliability. Randomly divide the benchmark items into two halves, score each half separately, and correlate the half-scores. The Spearman-Brown correction adjusts for the fact that each half is only half as long as the full test:

$$\rho_{XX'} = \frac{2r_{12}}{1 + r_{12}}$$

where r_{12} is the correlation between the two half-scores. For AI evaluation, this is easy to compute from existing data—no re-running required.

Internal consistency (Cronbach's alpha). The most widely used reliability coefficient generalizes split-half reliability by averaging over all possible splits:

$$\alpha = \frac{M}{M-1}\left(1 - \frac{\sum_{j=1}^{M}\sigma_j^2}{\sigma_X^2}\right) \tag{5.1}$$

where M is the number of items, σ_j^2 is the variance of item j across models, and σ_X^2 is the variance of total scores.

What Cronbach's Alpha Does and Does Not Tell You

Alpha measures how much items covary relative to total variance. High alpha means the items are measuring something in common. But alpha does *not* tell you: (a) whether that common thing is unidimensional, (b) whether that thing is the construct you intended, 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.

The Spearman-Brown prophecy formula predicts how reliability changes with test length. If a test of M items has reliability ρ, a test of KM items (made of K parallel forms) has reliability:

$$\rho_K = \frac{K\rho}{1 + (K-1)\rho} \tag{5.2}$$

This formula is practically important: it tells you how many items you need to achieve a target reliability. If your 50-item benchmark has $\rho = 0.80$, you need $K = 2$ (100 items) to reach $\rho = 0.89$, or $K = 4$ (200 items) to reach $\rho = 0.94$.

```python
#| 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,
})
```

```python
#| label: reliability-coefficients
#| autorun: true
#| fig-cap: "Left: Split-half reliability from a simulated Rasch benchmark. Center:
 ↪  Cronbach's alpha increases with test length following the Spearman-Brown prophecy.
 ↪  Right: SEM decreases as reliability improves with more items."

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import pearsonr

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

# Simulate benchmark data under the Rasch model
np.random.seed(42)
N = 100  # models
M = 100  # items
theta = np.random.normal(0, 1, N)
beta = np.random.normal(0, 1.5, M)

# Generate binary response matrix
P = sigmoid(theta[:, None] - beta[None, :])
Y = (np.random.rand(N, M) < P).astype(float)

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

# Panel 1: Split-half reliability
half1 = Y[:, :M//2].mean(axis=1)
half2 = Y[:, M//2:].mean(axis=1)
r12, _ = pearsonr(half1, half2)
rho_sb = 2 * r12 / (1 + r12)

axes[0].scatter(half1, half2, alpha=0.5, s=15, color='steelblue')
axes[0].plot([0, 1], [0, 1], 'k--', alpha=0.3)
axes[0].set_xlabel('Half 1 Score')
axes[0].set_ylabel('Half 2 Score')
axes[0].set_title(f'Split-Half (  = {rho_sb:.3f})')
axes[0].grid(True, alpha=0.3)

# Panel 2: Cronbach's alpha vs number of items (Spearman-Brown)
item_counts = np.arange(5, M + 1, 5)
alphas = []
for m in item_counts:
    Y_sub = Y[:, :m]
    item_vars = Y_sub.var(axis=0, ddof=1)
    total_var = Y_sub.sum(axis=1).var(ddof=1)
    alpha_val = (m / (m - 1)) * (1 - item_vars.sum() / total_var)
    alphas.append(alpha_val)

# Spearman-Brown prediction from base reliability (5 items)
base_rho = alphas[0]
K_vals = item_counts / item_counts[0]
sb_predicted = K_vals * base_rho / (1 + (K_vals - 1) * base_rho)

axes[1].plot(item_counts, alphas, 'o-', color='steelblue', markersize=3,
             linewidth=1.5, label='Empirical  ')
axes[1].plot(item_counts, sb_predicted, '--', color='coral', linewidth=1.5,
             label='Spearman-Brown')
axes[1].set_xlabel('Number of Items')
axes[1].set_ylabel("Cronbach's  ")
axes[1].set_title('Reliability vs Test Length')
axes[1].legend(fontsize=7)
axes[1].grid(True, alpha=0.3)
axes[1].set_ylim(0, 1)

# Panel 3: SEM vs number of items
sigma_X = np.array([Y[:, :m].mean(axis=1).std() for m in item_counts])
sems = sigma_X * np.sqrt(1 - np.array(alphas))

axes[2].plot(item_counts, sems, 'o-', color='steelblue', markersize=3, linewidth=1.5)
axes[2].set_xlabel('Number of Items')
axes[2].set_ylabel('SEM (score units)')
axes[2].set_title('Standard Error of Measurement')
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"Full test ({M} items):   = {alphas[-1]:.3f}, SEM = {sems[-1]:.4f}")
print(f"Half test ({M//2} items):   = {alphas[len(alphas)//2 - 1]:.3f}")
print(f"Spearman-Brown from 5 items: predicted   at {M} items =
 ↪  {sb_predicted[-1]:.3f}")
```

5.3 *Sources of Noise in AI Evaluation*

Classical Test Theory lumps all error into a single term E. To improve evaluation reliability, we need to understand *where* the noise comes from. This section provides a taxonomy of noise sources specific to AI evaluation.

5.3.1 *A Taxonomy of Error Sources*

We organize noise sources by the level at which they operate:

Source	Level	CTT Interpretation	Example
Sampling stochasticity	Within-model	Test-retest error	Different outputs at temperature > 0
Prompt sensitivity	Within-model	Format effect	Different prompt templates yield different scores
Item sampling	Within-benchmark	Parallel forms error	Different subsets of questions
Annotator disagreement	Within-scoring	Rater error	Different humans score the same output differently
LLM judge variability	Within-scoring	Rater error	Different LLM judges disagree
Rubric ambiguity	Within-scoring	Systematic + random	Vague criteria interpreted differently
API/version drift	Across-time	Test-retest error	Model weights updated silently

Each source contributes to σ_E^2, but they do so in different ways and require different remedies. Prompt sensitivity and annotator disagreement may be the dominant sources for open-ended evaluation, while item sampling variability dominates for fixed-format benchmarks.

5.3.2 *Prompt Sensitivity*

Even deterministic models (temperature = 0) produce different responses under different prompt formats. Adding "Let's think step by step" to a reasoning prompt, changing the answer format from multiple choice to open-ended, or even rearranging the few-shot examples can shift model rankings (Mizrahi et al. 2024). This is analogous to *test format effects* in educational testing—the same knowledge is assessed differently by a multiple-choice exam versus a free-response exam.

Prompt sensitivity threatens reliability because the "true score" depends on which prompt template is used. If model A outperforms model B on one template but not another, the models' relative standing is prompt-dependent, not ability-dependent.

5.3.3 *Sampling Stochasticity*

When temperature > 0, the same model gives different outputs to the same prompt on each run. This is the closest analogue to classical test-retest error: the model's response is a random draw from its distribution, and different draws yield different scores. The within-model variance across runs directly inflates σ_E^2.

For binary items (correct/incorrect), this source is minimal—most models give the same answer deterministically. But for open-ended generation (summaries, code, essays), stochastic outputs can receive different quality scores across runs.

5.3.4 *Annotator and Judge Variability*

For tasks where responses cannot be automatically scored—creative writing, safety evaluation, open-ended reasoning—human annotators or LLM judges must assess quality. Different raters often disagree, and this disagreement is a major source of unreliability.

Definition: Cohen's Kappa

For two raters scoring the same set of items on a categorical scale, Cohen's kappa (Cohen 1960) adjusts for chance agreement:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

where p_o is the observed proportion of agreement and p_e is the expected proportion of agreement by chance. $\kappa = 1$ indicates perfect agreement; $\kappa = 0$ indicates agreement no better than chance; $\kappa < 0$ indicates systematic disagreement.

For more than two raters, Fleiss's kappa (Fleiss 1971) extends this to multiple annotators. Krippendorff's alpha (Krippendorff 2011) provides a more general measure that handles missing data, ordinal scales, and any number of raters.

5.3.5 *Benchmark Sampling Variability*

The benchmark itself is a sample from a larger domain of possible items. Different samples yield different model rankings. A 200-item coding benchmark is a sample from the space of all possible coding problems; another sample of 200 problems from the same domain would likely produce somewhat different scores.

This is the item sampling component of reliability, and it is the source that Equation 5.2 addresses: longer tests sample the domain more thoroughly and are therefore more reliable. It also connects to Chapter 3's design perspective: D-optimal item selection minimizes this variability by choosing a maximally informative sample.

```
#| label: noise-decomposition
#| autorun: true
#| fig-cap: "Left: Variance decomposition reveals that item sampling and annotator
  disagreement typically dominate. Center: Model rankings shift across evaluation
  conditions. Right: Overlapping confidence intervals show where rankings are
  unreliable."

```

```
np.random.seed(42)
N = 20    # models
M = 50    # items
R = 5     # raters per item

# True abilities and item difficulties
theta = np.linspace(-2, 2, N)
beta = np.random.normal(0, 1.5, M)

# Variance parameters for each noise source
sigma_prompt = 0.3    # prompt sensitivity (shifts difficulty)
sigma_sample = 0.0    # item sampling (captured by different subsets)
sigma_rater = 0.5     # rater noise (scoring variability)
sigma_stoch = 0.2     # sampling stochasticity (model randomness)

# Generate a full evaluation with all noise sources
def run_evaluation(theta, beta, sigma_prompt, sigma_rater, sigma_stoch, seed=None):
    if seed is not None:
        np.random.seed(seed)
    N, M = len(theta), len(beta)
    # Prompt effect: shifts item difficulties
    prompt_shift = np.random.normal(0, sigma_prompt, M)
    beta_eff = beta + prompt_shift
    # Model stochasticity: shifts ability
    stoch_shift = np.random.normal(0, sigma_stoch, N)
    theta_eff = theta + stoch_shift
    # Generate responses
    P = sigmoid(theta_eff[:, None] - beta_eff[None, :])
    Y = (np.random.rand(N, M) < P).astype(float)
    # Rater noise on total scores
    scores = Y.mean(axis=1) + np.random.normal(0, sigma_rater / np.sqrt(M), N)
    return scores, Y

# Run multiple replications
n_reps = 200
all_scores = np.zeros((n_reps, N))
for rep in range(n_reps):
    scores, _ = run_evaluation(theta, beta, sigma_prompt, sigma_rater, sigma_stoch,
 ↪  seed=rep)
    all_scores[rep] = scores

# Variance decomposition (approximate via simulation)
mean_scores = all_scores.mean(axis=0)
total_var = all_scores.var()
between_model_var = mean_scores.var()
within_model_var = all_scores.var(axis=0).mean()

# Run with individual sources turned off to estimate contributions
var_components = {}
```

```
for name, sp, sr, ss in [('Prompt', 0, sigma_rater, sigma_stoch),
                          ('Rater', sigma_prompt, 0, sigma_stoch),
                          ('Stochastic', sigma_prompt, sigma_rater, 0),
                          ('Full', sigma_prompt, sigma_rater, sigma_stoch)]:
    reps = np.zeros((100, N))
    for rep in range(100):
        s, _ = run_evaluation(theta, beta, sp, sr, ss, seed=rep + 1000)
        reps[rep] = s
    var_components[name] = reps.var(axis=0).mean()

# Estimate each component's contribution
prompt_contrib = var_components['Full'] - var_components['Prompt']
rater_contrib = var_components['Full'] - var_components['Rater']
stoch_contrib = var_components['Full'] - var_components['Stochastic']
residual = max(0, within_model_var - prompt_contrib - rater_contrib - stoch_contrib)

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

# Panel 1: Variance decomposition
components = [between_model_var, prompt_contrib, rater_contrib, stoch_contrib]
labels = ['Model\n(signal)', 'Prompt\nformat', 'Rater\nnoise', 'Sampling\nstoch.']
colors = ['#2ca02c', '#ff7f0e', '#d62728', '#9467bd']
bars = axes[0].bar(labels, components, color=colors, alpha=0.8)
axes[0].set_ylabel('Variance')
axes[0].set_title('Variance Decomposition')
axes[0].grid(True, alpha=0.3, axis='y')

# Panel 2: Ranking shifts across conditions
conditions = [run_evaluation(theta, beta, sigma_prompt, sigma_rater, sigma_stoch,
 ↪  seed=s)
              for s in [10, 20, 30]]
rankings = [np.argsort(-s[0]) for s in conditions]

for i in range(N):
    positions = [np.where(r == i)[0][0] for r in rankings]
    color = plt.cm.viridis(theta[i] / 4 + 0.5)
    axes[1].plot([1, 2, 3], positions, '-o', color=color, alpha=0.5,
                 markersize=3, linewidth=0.8)

axes[1].set_xlabel('Evaluation Run')
axes[1].set_ylabel('Rank')
axes[1].set_title('Ranking Instability')
axes[1].set_xticks([1, 2, 3])
axes[1].invert_yaxis()
axes[1].grid(True, alpha=0.3)

# Panel 3: Score distributions with confidence intervals
score_means = all_scores.mean(axis=0)
score_sems = all_scores.std(axis=0)
```

```
order = np.argsort(score_means)

axes[2].barh(range(N), score_means[order], xerr=1.96 * score_sems[order],
             color='steelblue', alpha=0.7, capsize=2, height=0.7)
axes[2].set_xlabel('Mean Score')
axes[2].set_ylabel('Model (ranked)')
axes[2].set_title('Scores with 95% CI')
axes[2].set_yticks([])
axes[2].grid(True, alpha=0.3, axis='x')

plt.tight_layout()
plt.show()

reliability = between_model_var / total_var
print(f"Signal (between-model) variance: {between_model_var:.4f}")
print(f"Noise (within-model) variance: {within_model_var:.4f}")
print(f"Estimated reliability: {reliability:.3f}")
```

5.4 Generalizability Theory

5.4.1 From One Error to Many

CTT gives a single reliability number, collapsing all sources of error into σ_E^2. This is useful for a quick summary but insufficient when you need to *improve* reliability—you cannot fix what you have not diagnosed. If your evaluation is unreliable because annotators disagree, adding more items will not help. If it is unreliable because of item sampling variability, adding more annotators will not help.

Generalizability Theory (G-theory), developed by Cronbach et al. (1972), addresses this limitation by modeling multiple *facets* of measurement simultaneously. Where CTT asks "how reliable is this test?", G-theory asks "how much variance is attributable to each source, and how can we design a measurement procedure that minimizes error?"

5.4.2 The G-Theory Framework

Consider an evaluation where N models are tested on M items, each scored by R raters. The observed score for model p, item i, rater r is decomposed as:

$$X_{pir} = \mu + \alpha_p + \beta_i + \gamma_r + (\alpha\beta)_{pi} + (\alpha\gamma)_{pr} + (\beta\gamma)_{ir} + \epsilon_{pir}$$

where μ is the grand mean and each Greek letter represents a random effect:

- α_p: model effect (the signal we want to measure), variance σ_p^2
- β_i: item effect (some items are harder), variance σ_i^2
- γ_r: rater effect (some raters are lenient), variance σ_r^2

- $(\alpha\beta)_{pi}$: model-by-item interaction (item i is especially hard for model p), variance σ^2_{pi}
- $(\alpha\gamma)_{pr}$: model-by-rater interaction (rater r is especially harsh on model p), variance σ^2_{pr}
- $(\beta\gamma)_{ir}$: item-by-rater interaction, variance σ^2_{ir}
- ϵ_{pir}: residual (everything else), variance $\sigma^2_{pir,e}$

The AI evaluation translation is direct:

G-theory term	AI evaluation analogue
Person (p)	Model being evaluated
Item (i)	Benchmark question or task
Rater (r)	Human annotator or LLM judge
Occasion (o)	Run, random seed, or prompt variant

Definition: Generalizability Coefficient

The generalizability coefficient is the ratio of true score variance to true-plus-relative-error variance:

$$G = \frac{\sigma^2_p}{\sigma^2_p + \sigma^2_{\text{rel}}}$$

where the relative error variance for a design with n_i items and n_r raters is:

$$\sigma^2_{\text{rel}} = \frac{\sigma^2_{pi}}{n_i} + \frac{\sigma^2_{pr}}{n_r} + \frac{\sigma^2_{pir,e}}{n_i \cdot n_r}$$

5.4.3 G-Studies and D-Studies

G-theory separates the estimation and design phases:

G-study (Generalizability study): Collect data from a fully or partially crossed design and estimate all variance components. This tells you *where the noise is*—the diagnostic step. Variance components are typically estimated using the method of expected mean squares from ANOVA or via restricted maximum likelihood (REML).

D-study (Decision study): Given the variance components from a G-study, compute the generalizability coefficient for a *planned* evaluation design. This answers questions like: “How many items and raters do I need to achieve $G \geq 0.90$?”

The key insight is that you can *trade off* facets against each other. If rater variance dominates (σ^2_{pr} is large), increasing n_r is more cost-effective than increasing n_i. If item sampling variance dominates (σ^2_{pi} is large), more items help more than more raters. The D-study makes these tradeoffs explicit and quantitative.

```
#| label: gtheory-simulation
#| autorun: true
#| fig-cap: "Left: Variance component estimates from a simulated G-study reveal where
 ↪  noise originates. Center: D-study shows how G-coefficient improves with more items
 ↪  and raters. Right: Cost-optimal designs for different budgets."

np.random.seed(42)

# True variance components
true_var = {
    'p': 1.0,     # model (signal)
    'i': 0.5,     # item
    'r': 0.3,     # rater
    'pi': 0.4,    # model x item interaction
    'pr': 0.3,    # model x rater interaction
    'ir': 0.1,    # item x rater interaction
    'e': 0.2      # residual
}

# Simulate data: 30 models x 20 items x 5 raters
N_p, N_i, N_r = 30, 20, 5

# Generate random effects
alpha_p = np.random.normal(0, np.sqrt(true_var['p']), N_p)
beta_i = np.random.normal(0, np.sqrt(true_var['i']), N_i)
gamma_r = np.random.normal(0, np.sqrt(true_var['r']), N_r)
ab_pi = np.random.normal(0, np.sqrt(true_var['pi']), (N_p, N_i))
ag_pr = np.random.normal(0, np.sqrt(true_var['pr']), (N_p, N_r))
bg_ir = np.random.normal(0, np.sqrt(true_var['ir']), (N_i, N_r))
eps = np.random.normal(0, np.sqrt(true_var['e']), (N_p, N_i, N_r))

# Construct observed scores
mu = 5.0
X = (mu + alpha_p[:, None, None] + beta_i[None, :, None] + gamma_r[None, None, :]
     + ab_pi[:, :, None] + ag_pr[:, None, :] + bg_ir[None, :, :] + eps)

# Estimate variance components via method of moments (ANOVA)
grand_mean = X.mean()
MS_p = N_i * N_r * X.mean(axis=(1, 2)).var(ddof=1)
MS_i = N_p * N_r * X.mean(axis=(0, 2)).var(ddof=1)
MS_r = N_p * N_i * X.mean(axis=(0, 1)).var(ddof=1)
MS_pi = N_r * X.mean(axis=2).var(ddof=1) * N_p * N_i / ((N_p - 1) * (N_i - 1))
MS_pr = N_i * X.mean(axis=1).var(ddof=1) * N_p * N_r / ((N_p - 1) * (N_r - 1))

# Simplified variance component estimates
est_var_p = max(0, (MS_p - MS_pi) / (N_i * N_r))
est_var_i = max(0, (MS_i) / (N_p * N_r))
est_var_r = max(0, (MS_r) / (N_p * N_i))

```

```
# For interactions, use residual-based estimates
X_pi = X.mean(axis=2)  # average over raters
est_var_pi = max(0, X_pi.var() - est_var_p - est_var_i - true_var['e'] / N_r)
X_pr = X.mean(axis=1)  # average over items
est_var_pr = max(0, X_pr.var() - est_var_p - est_var_r - true_var['e'] / N_i)
est_var_e = max(0.01, X.var() - est_var_p - est_var_i - est_var_r
                - est_var_pi - est_var_pr)

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

# Panel 1: Variance components
comp_names = ['Model\n(signal)', 'Item', 'Rater', 'Model×\nItem', 'Model×\nRater',
 ↪  'Residual']
true_vals = [true_var['p'], true_var['i'], true_var['r'],
             true_var['pi'], true_var['pr'], true_var['e']]
colors = ['#2ca02c', '#1f77b4', '#d62728', '#ff7f0e', '#9467bd', '#8c564b']
bars = axes[0].bar(comp_names, true_vals, color=colors, alpha=0.8)
axes[0].set_ylabel('Variance')
axes[0].set_title('Variance Components')
axes[0].grid(True, alpha=0.3, axis='y')
axes[0].tick_params(axis='x', rotation=30)

# Panel 2: D-study surface
n_items_range = np.arange(5, 51, 5)
n_raters_range = np.arange(1, 11)

G_matrix = np.zeros((len(n_raters_range), len(n_items_range)))
for ii, ni in enumerate(n_items_range):
    for ri, nr in enumerate(n_raters_range):
        rel_error = (true_var['pi'] / ni + true_var['pr'] / nr
                     + true_var['e'] / (ni * nr))
        G_matrix[ri, ii] = true_var['p'] / (true_var['p'] + rel_error)

im = axes[1].contourf(n_items_range, n_raters_range, G_matrix,
                       levels=np.arange(0.5, 1.01, 0.05), cmap='RdYlGn')
axes[1].contour(n_items_range, n_raters_range, G_matrix,
                levels=[0.80, 0.90], colors='black', linewidths=1.5)
axes[1].set_xlabel('Number of Items')
axes[1].set_ylabel('Number of Raters')
axes[1].set_title('G-Coefficient (D-Study)')
plt.colorbar(im, ax=axes[1], shrink=0.8)

# Panel 3: Cost-optimal designs
cost_item = 1.0   # cost per item (e.g., API call)
cost_rater = 5.0  # cost per rater (e.g., annotation)

budgets = np.arange(50, 501, 10)
optimal_G = []
optimal_ni = []
```

```
optimal_nr = []

for budget in budgets:
    best_G = 0
    best_ni, best_nr = 5, 1
    for ni in range(5, int(budget / cost_item) + 1):
        remaining = budget - ni * cost_item
        nr = max(1, int(remaining / (ni * cost_rater)))
        if nr < 1:
            continue
        total_cost = ni * cost_item + ni * nr * cost_rater
        if total_cost > budget:
            nr = max(1, nr - 1)
        rel_error = (true_var['pi'] / ni + true_var['pr'] / nr
                     + true_var['e'] / (ni * nr))
        G = true_var['p'] / (true_var['p'] + rel_error)
        if G > best_G:
            best_G = G
            best_ni, best_nr = ni, nr
    optimal_G.append(best_G)
    optimal_ni.append(best_ni)
    optimal_nr.append(best_nr)

axes[2].plot(budgets, optimal_G, 'steelblue', linewidth=1.5)
axes[2].axhline(0.90, color='red', linestyle='--', linewidth=1, label='G = 0.90')
axes[2].set_xlabel('Budget')
axes[2].set_ylabel('Optimal G-Coefficient')
axes[2].set_title('Cost-Optimal Design')
axes[2].legend(fontsize=7)
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Find minimum budget for G >= 0.90
for b, g in zip(budgets, optimal_G):
    if g >= 0.90:
        print(f"Minimum budget for G   0.90: {b}
 ↪  (items={optimal_ni[list(budgets).index(b)]}, "
              f"raters={optimal_nr[list(budgets).index(b)]})")
        break
```

5.5 *LLM-as-a-Judge Reliability*

5.5.1 *The Rise of Automated Scoring*

As AI systems are increasingly evaluated on open-ended tasks—summarization, creative writing, instruction following, safety—human annotation becomes the bottleneck. LLMs are now widely used as judges, scoring model outputs on rubric-based criteria or making pairwise preference judgments (Zheng et al. 2023). This creates a new measurement instrument whose reliability properties must be understood.

In the G-theory framework, an LLM judge is a *rater*. It has its own biases (γ_r), interacts differently with different models ($\alpha\gamma_{pr}$), and may score the same output differently on repeated calls (if temperature > 0). The reliability analysis from the previous section applies directly, with the LLM judge playing the role of the human annotator.

5.5.2 Inter-Rater Agreement

Three questions characterize LLM-as-a-judge reliability:

1. **Inter-judge reliability**: Do different LLM judges agree with each other? If GPT-4, Claude, and Gemini all judge the same outputs, do they produce the same rankings?
2. **Human-LLM agreement**: Does the LLM judge agree with human annotators? High agreement suggests the LLM captures human preferences; low agreement means the LLM measures something different.
3. **Intra-judge consistency**: Does the same LLM judge produce the same score when asked twice? At temperature 0 this should be deterministic, but prompt-order effects and other sources of instability can create variability.

5.5.3 Position Bias and Systematic Error

LLM judges exhibit position bias: in pairwise comparisons, they tend to prefer the response presented first (or last, depending on the model) (P. Wang et al. 2023). This is a *systematic* error, not random noise.

! Systematic vs. Random Error

CTT's error term E captures *random* error: zero mean, uncorrelated across occasions. *Systematic* errors—like position bias or a judge that always prefers longer responses—inflate the true score, not the error term. A biased judge can be highly reliable (consistent in its bias) yet invalid.

Reliability analysis detects random error. Detecting systematic error requires the validity tools from Chapter 6. A high inter-rater agreement between two LLM judges does not mean they are correct—they might share the same systematic bias.

5.5.4 Designing Reliable Judge Protocols

Based on the reliability framework, practical strategies for improving LLM-as-a-judge reliability include:

1. **Multiple judges with aggregation**. Using $n_r > 1$ judges and taking the majority vote or average score reduces error variance by a factor of $1/n_r$ (the G-theory D-study). Three independent judges are substantially more reliable than one.
2. **Position randomization**. Present responses in both orders and average the judgments. This eliminates position bias, converting a systematic error into cancelled noise.
3. **Deterministic scoring**. Use temperature 0 to eliminate sampling stochasticity. This removes one source of σ_E^2 entirely.
4. **Rubric specificity**. Vague rubrics like "which response is better?" produce more disagreement than specific criteria like "which response correctly follows the instruction and provides accurate information?" Specific rubrics reduce σ_r^2 and σ_{ir}^2.
5. **Calibration against gold standards**. Before deploying an LLM judge, measure its agreement with human annotations on a calibration set. This provides an estimate of systematic bias (not just random error).

```python
#| label: llm-judge-reliability
#| autorun: true
#| fig-cap: "Left: Inter-judge agreement matrix (Cohen's kappa) for simulated LLM
↪  judges. Center: Accuracy improves with majority voting across more judges. Right:
↪  Position bias is a systematic error that reduces validity but not reliability."

np.random.seed(42)

N_pairs = 200  # pairwise comparisons
N_judges = 5

# True quality differences (positive = first response is better)
true_diff = np.random.normal(0, 1, N_pairs)
true_labels = (true_diff > 0).astype(int)

# Simulate judges with different quality levels and biases
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):
    # Each judge: correct with probability quality, plus position bias
    p_correct = judge_quality[j]
    for k in range(N_pairs):
        if np.random.rand() < p_correct:
            judgments[j, k] = true_labels[k]
        else:
            judgments[j, k] = 1 - true_labels[k]
        # Position bias: tendency to choose first response
        if np.random.rand() < abs(judge_bias[j]):
            judgments[j, k] = 1 if judge_bias[j] > 0 else 0
```

```

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

# Panel 1: Inter-judge kappa matrix
kappa_matrix = np.zeros((N_judges, N_judges))
for j1 in range(N_judges):
    for j2 in range(N_judges):
        if j1 == j2:
            kappa_matrix[j1, j2] = 1.0
        else:
            # Compute Cohen's kappa
            p_o = np.mean(judgments[j1] == judgments[j2])
            p1_yes = judgments[j1].mean()
            p2_yes = judgments[j2].mean()
            p_e = p1_yes * p2_yes + (1 - p1_yes) * (1 - p2_yes)
            kappa_matrix[j1, j2] = (p_o - p_e) / (1 - p_e) if p_e < 1 else 0

im = axes[0].imshow(kappa_matrix, cmap='RdYlGn', vmin=0, vmax=1)
axes[0].set_xticks(range(N_judges))
axes[0].set_yticks(range(N_judges))
axes[0].set_xticklabels([f'J{i+1}' for i in range(N_judges)])
axes[0].set_yticklabels([f'J{i+1}' for i in range(N_judges)])
axes[0].set_title("Inter-Judge  ")
for i in range(N_judges):
    for j in range(N_judges):
        axes[0].text(j, i, f'{kappa_matrix[i,j]:.2f}', ha='center', va='center',
 ↪  fontsize=6)
plt.colorbar(im, ax=axes[0], shrink=0.8)

# Panel 2: Accuracy vs number of judges (majority vote)
judge_counts = range(1, N_judges + 1)
accuracies = []
for n_j in judge_counts:
    # Average over many random subsets of judges
    acc_samples = []
    for _ in range(100):
        subset = np.random.choice(N_judges, n_j, replace=False)
        majority = (judgments[subset].mean(axis=0) > 0.5).astype(int)
        acc_samples.append(np.mean(majority == true_labels))
    accuracies.append(np.mean(acc_samples))

axes[1].plot(list(judge_counts), accuracies, 'o-', color='steelblue',
             linewidth=1.5, markersize=5)
axes[1].set_xlabel('Number of Judges')
axes[1].set_ylabel('Accuracy')
axes[1].set_title('Majority Vote Accuracy')
axes[1].grid(True, alpha=0.3)
axes[1].set_xticks(list(judge_counts))

```

```
# Panel 3: Effect of position bias on accuracy
bias_levels = np.linspace(0, 0.3, 20)
acc_with_bias = []
acc_debiased = []

for bias in bias_levels:
    # Single judge with varying bias
    correct = np.zeros(N_pairs)
    biased = np.zeros(N_pairs)
    for k in range(N_pairs):
        # Without debiasing
        if np.random.rand() < 0.80:
            pred = true_labels[k]
        else:
            pred = 1 - true_labels[k]
        if np.random.rand() < bias:
            pred = 1  # always pick first
        biased[k] = pred

        # With debiasing (average both orders)
        pred1, pred2 = pred, pred
        if np.random.rand() < bias:
            pred2 = 0  # reversed order: pick "first" = second
        correct[k] = round((pred1 + (1 - pred2)) / 2)

    acc_with_bias.append(np.mean(biased == true_labels))
    acc_debiased.append(np.mean(correct == true_labels))

axes[2].plot(bias_levels, acc_with_bias, 'r-', linewidth=1.5, label='With bias')
axes[2].plot(bias_levels, acc_debiased, 'g-', linewidth=1.5, label='Debiased')
axes[2].set_xlabel('Position Bias Strength')
axes[2].set_ylabel('Accuracy')
axes[2].set_title('Position Bias Effect')
axes[2].legend(fontsize=7)
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

mean_kappa = kappa_matrix[np.triu_indices(N_judges, k=1)].mean()
print(f"Mean inter-judge  : {mean_kappa:.3f}")
print(f"Single judge accuracy: {accuracies[0]:.3f}")
print(f"5-judge majority accuracy: {accuracies[-1]:.3f}")
```

5.6 Reliability Under IRT

5.6.1 Conditional Reliability

CTT provides a single reliability number for the entire test. But from Chapter 2 and Chapter 4, we know that measurement precision varies across the ability range. A benchmark designed for mid-range models may be highly reliable at $\theta = 0$ but unreliable at the extremes.

Under IRT, the precision of measurement at ability θ is captured by the test information function (introduced in Section 4.2.1):

$$I(\theta) = \sum_{j=1}^{M} P_j(\theta)(1 - P_j(\theta))$$

The *conditional reliability* at θ is:

$$\rho(\theta) = 1 - \frac{1}{I(\theta) \cdot \sigma_\theta^2}$$

where σ_θ^2 is the variance of abilities in the population. This formula makes a key point: reliability is not a fixed property of the test—it depends on *where* in the ability range you are measuring and *how* spread out the population is.

5.6.2 The Spearman-Brown Prophecy Under IRT

The classical Spearman-Brown formula (Equation 5.2) says reliability increases with test length. Under IRT, this is approximately true when items are well-targeted to the ability range of interest, but breaks down when items are poorly matched.

Adding 50 easy items to a benchmark that already has 50 hard items does little to improve reliability for high-ability models—the easy items provide almost no information in that range. This is precisely the insight behind adaptive testing (Section 4.2.2): by selecting items matched to each model's ability, CAT achieves the same reliability with fewer items.

```
#| label: conditional-reliability
#| autorun: true
#| fig-cap: "Left: Test information function for different item pool designs. Center:
↪  Conditional reliability varies across the ability range. Right: Adaptive testing
↪  achieves target reliability with fewer items at all ability levels."

np.random.seed(42)

theta_grid = np.linspace(-3, 3, 200)
sigma_theta = 1.0  # population SD

```

```python
# Three item pool designs
M = 30
# Design 1: Uniform spread
beta_uniform = np.linspace(-2, 2, M)
# Design 2: Concentrated at center
beta_center = np.random.normal(0, 0.5, M)
# Design 3: Concentrated at extremes (bimodal)
beta_extreme = np.concatenate([np.random.normal(-1.5, 0.3, M//2),
                                np.random.normal(1.5, 0.3, M//2)])

def test_info(theta_grid, betas):
    info = np.zeros_like(theta_grid)
    for b in betas:
        P = sigmoid(theta_grid - b)
        info += P * (1 - P)
    return info

def cond_reliability(info, sigma_sq):
    return 1 - 1 / (info * sigma_sq + 1e-10)

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

# Panel 1: Test information functions
designs = [('Uniform', beta_uniform, '#2ca02c'),
           ('Centered', beta_center, '#1f77b4'),
           ('Extreme', beta_extreme, '#d62728')]

for name, betas, color in designs:
    info = test_info(theta_grid, betas)
    axes[0].plot(theta_grid, info, color=color, linewidth=1.5, label=name)

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)

# Panel 2: Conditional reliability
for name, betas, color in designs:
    info = test_info(theta_grid, betas)
    rho = np.clip(cond_reliability(info, sigma_theta**2), 0, 1)
    axes[1].plot(theta_grid, rho, color=color, linewidth=1.5, label=name)

axes[1].axhline(0.90, color='black', linestyle='--', linewidth=1, alpha=0.5)
axes[1].set_xlabel('Ability ( )')
axes[1].set_ylabel('Conditional Reliability')
axes[1].set_title('Reliability Across Ability')
axes[1].legend(fontsize=7)
axes[1].grid(True, alpha=0.3)
```

```python
axes[1].set_ylim(0, 1)

# Panel 3: Items needed to reach target reliability
target_rho = 0.90
theta_test_points = np.linspace(-2.5, 2.5, 50)

items_needed_uniform = []
items_needed_adaptive = []

for theta_t in theta_test_points:
    # Uniform: add items from uniform pool one at a time
    betas_sorted = sorted(beta_uniform, key=lambda b: abs(b - theta_t))
    for k in range(1, M + 1):
        info = sum(sigmoid(theta_t - b) * (1 - sigmoid(theta_t - b)) for b in betas_sorted[:k])
        rho = 1 - 1 / (info * sigma_theta**2 + 1e-10)
        if rho >= target_rho:
            items_needed_uniform.append(k)
            break
    else:
        items_needed_uniform.append(M)

    # Adaptive: always pick item closest to theta
    all_betas = np.random.normal(0, 1.5, 100)
    betas_adapt = sorted(all_betas, key=lambda b: abs(b - theta_t))
    for k in range(1, len(betas_adapt) + 1):
        info = sum(sigmoid(theta_t - b) * (1 - sigmoid(theta_t - b)) for b in betas_adapt[:k])
        rho = 1 - 1 / (info * sigma_theta**2 + 1e-10)
        if rho >= target_rho:
            items_needed_adaptive.append(k)
            break
    else:
        items_needed_adaptive.append(len(betas_adapt))

axes[2].plot(theta_test_points, items_needed_uniform, color='#1f77b4',
             linewidth=1.5, label='Fixed (uniform)')
axes[2].plot(theta_test_points, items_needed_adaptive, color='#2ca02c',
             linewidth=1.5, label='Adaptive')
axes[2].set_xlabel('Ability ( )')
axes[2].set_ylabel('Items Needed')
axes[2].set_title(f'Items for    {target_rho}')
axes[2].legend(fontsize=7)
axes[2].grid(True, alpha=0.3)

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

The practical implication is clear: a single reliability number can be misleading. A benchmark might report $\alpha = 0.92$ overall, but have conditional reliability below 0.80 for the frontier models that matter most—because all the items are too easy for them. The IRT framework provides the tools to diagnose and fix this problem, connecting directly to the efficient measurement methods of Chapter 4.

5.7 *Designing Reliable AI Evaluations*

The theory developed in this chapter yields concrete design principles for AI evaluation:

1. **Know your dominant noise source**. Run a G-study before committing to an evaluation design. If annotator variability dominates, invest in more raters or clearer rubrics. If item sampling variability dominates, use more items or D-optimal item selection.

2. **Match items to the ability range of interest**. Conditional reliability shows that measurement precision varies across ability levels. If the goal is to rank frontier models, the benchmark needs items that are difficult enough to discriminate among them.

3. **Use multiple judges when scoring is subjective**. The G-theory D-study quantifies the improvement from adding judges. Three independent judges with majority vote is substantially more reliable than a single judge, and often more cost-effective than adding more items.

4. **Report confidence intervals, not just point estimates**. The SEM provides a natural uncertainty band. Two models whose scores overlap within $\pm 2 \times$ SEM may not be meaningfully different. Leaderboards should indicate which ranking differences are statistically reliable.

5. **Control what you can**. Use temperature 0 for deterministic scoring. Randomize prompt formats and average across them. Standardize rubrics. These measures eliminate sources of σ_E^2 rather than averaging over them.

6. **Separate reliability from validity**. High reliability is necessary but not sufficient. A benchmark can be highly reliable while systematically measuring the wrong thing (contamination, shortcut features, length bias). The tools for detecting these problems are in Chapter 6.

7. **Audit items using reliability diagnostics**. Individual items can degrade overall reliability. Items with negative item-total correlations *reduce* Cronbach's α — removing them improves reliability. S. T. Truong et al. (2025) demonstrate that this classical item analysis principle is a powerful tool for finding benchmark bugs: items with negative tetrachoric correlations or low Mokken scalability coefficients often have incorrect answer keys, ambiguous wording, or grading errors. On nine benchmarks, reliability-based flagging achieves up to 84% precision at the top-50 flagged items. The intuition is simple: under the Rasch model (or any unidimensional model), all items should correlate positively with each other and with the total score. Violations indicate items that are measuring something different from the rest of the benchmark — whether due to multidimensionality or outright errors.

5.8 Discussion Questions

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**. The Spearman-Brown formula predicts that longer tests are more reliable. AI benchmarks often have thousands of items—far more than typical educational tests. Does this mean AI benchmarks automatically have high reliability? What assumptions might be violated?

3. **LLM-as-a-judge tradeoffs**. Using GPT-4 as a judge is cheaper than human annotation but introduces model-specific biases. Under what conditions would you prefer a less accurate but more reliable judge? How does G-theory help formalize this tradeoff?

4. **Reliability across model generations**. When a new generation of models is released, the reliability of an existing benchmark may change (because item difficulties shift relative to model abilities). How should benchmark developers monitor and maintain reliability over time?

5. **The reliability-validity tension**. A benchmark consisting of 1000 copies of the same easy question would have extremely high internal consistency but near-zero validity. How does this pathological example illustrate the distinction between reliability and validity? What design principles prevent this?

6. **Item-level diagnostics and benchmark maintenance**. S. T. Truong et al. (2025) find that removing items with negative item-total correlations improves both reliability and validity of AI benchmarks. But removing items also shortens the test, which the Spearman-Brown formula predicts will *reduce* reliability. Under what conditions does the net effect of item removal improve reliability? How would you design an iterative item screening procedure that balances these competing pressures?

5.9 Bibliographic Notes

5.9.1 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).

5.9.2 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.

5.9.3 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).

5.9.4 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 P. 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.

5.9.5 Item-Level Reliability Diagnostics

Classical item analysis — computing item-total correlations, inter-item correlations, and Mokken scalability coefficients to identify misfitting items — is a standard step in educational test development. S. T. Truong et al. (2025) apply these techniques to AI benchmarks at scale, showing that items flagged by negative tetrachoric correlations or low scalability coefficients frequently contain genuine errors (incorrect keys, ambiguous wording, grading bugs). Their framework is grounded in the Rasch model's sufficiency property (Section 2.3.1): if sum scores are sufficient statistics for ability, all inter-item correlations must be non-negative, making violations a principled diagnostic signal. They recommend evaluating with 60–80 LLMs from at least 10 organizations for reliable detection.

5.10 Exercises

5.10.1 Theoretical Exercises

Exercise 4.1 ($\star$): Starting from the CTT model $X = T + E$ with the standard assumptions ($\mathbb{E}[E] = 0$, $\mathrm{Cov}(T, E) = 0$), derive that $\rho_{XX'} = \sigma_T^2/\sigma_X^2$. Show that reliability equals the squared correlation between observed and true scores: $\rho_{XX'} = r_{XT}^2$.

Exercise 4.2 ($\star\star$): Derive the Spearman-Brown prophecy formula. If a test of length M has reliability ρ, show that a test of length KM (made of K parallel forms) has reliability $\rho_K = K\rho/(1+(K-1)\rho)$. What happens as $K \to \infty$?

Exercise 4.3 ($\star\star$): In a G-theory $p \times i \times r$ design, derive the formula for the generalizability coefficient G. Show that increasing n_r reduces the contribution of rater variance but not item variance, and vice versa.

Exercise 4.4 ($\star\star$): Show that Cronbach's alpha is the mean of all possible split-half reliabilities (after Spearman-Brown correction) for a test with M items. Under what conditions is alpha a lower bound for reliability?

5.10.2 *Computational Exercises*

Exercise 4.5 ($\star\star$): Implement a G-study for a model $\times$ item $\times$ rater design. Given a 3-way data array $X[p, i, r]$, estimate all variance components using expected mean squares. Apply your implementation to simulated data and verify that the estimates recover the true variance components.

Exercise 4.6 ($\star\star$): Simulate an LLM-as-a-judge evaluation with 5 judges of varying quality (agreement with gold standard ranging from 0.70 to 0.90). Compute: (a) pairwise Cohen's kappa for all judge pairs, (b) accuracy with majority vote for 1, 3, and 5 judges. How does reliability change if all judges share the same systematic bias (e.g., preferring longer responses)?

Exercise 4.7 ($\star\star\star$): Implement a D-study optimizer. Given variance components and cost per item (c_i) and cost per rater-item pair (c_r), find the cheapest evaluation design (n_i, n_r) that achieves $G \geq G^*$. Solve this as a constrained optimization problem and visualize the cost surface.

5.10.3 *Discussion Exercises*

Exercise 4.8: The Chatbot Arena uses a single human preference judgment per comparison. Using G-theory, analyze the reliability of Elo ratings derived from this design. How many comparisons per model pair would be needed to achieve $G \geq 0.90$? What are the cost implications?

Exercise 4.9 ($\star\star$): Implement the item-level diagnostic pipeline from S. T. Truong et al. (2025). Simulate a Rasch benchmark with $N = 80$ models and $M = 200$ items. Introduce 20 "buggy" items: 10 with flipped answer keys (replace Y_{ij} with $1 - Y_{ij}$) and 10 with random responses (independent of θ). Compute: (a) item-total correlations, (b) average tetrachoric correlations, and (c) Mokken scalability coefficients H_j for all items. Rank items by each statistic. What is the precision@20 and recall@20 for each method? How does performance change as you vary the number of LLMs from 10 to 100?

6 Validity

Intended Learning Outcomes

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

1. **Define** validity in the context of AI evaluation and explain why it cannot be reduced to a single statistic.
2. **Distinguish** between content validity, criterion validity, construct validity, external validity, and consequential validity for AI benchmarks.
3. **Apply** Borsboom's realist framework and Salaudeen et al.'s claim-centered framework to evaluate whether a benchmark measures what it claims to measure.
4. **Identify** common threats to validity in AI evaluation: benchmark contamination, construct-irrelevant variance, construct underrepresentation, and differential item functioning.
5. **Use** diagnostic tools (DIF analysis, dimensionality assessment, item-fit statistics) to detect validity threats.
6. **Design** instrument construction and revision procedures, including the use of synthetic data for item generation.
7. **Evaluate** the validity of existing AI benchmarks using the frameworks presented in this chapter.

Suggested Lecture Plan

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

Lecture 1: Validity Frameworks and Evidence

– Borsboom's realist framework: validity as truth, warrant inference, semantic indeterminacy (20 min)
– From reliability to validity: motivating the distinction (10 min)
– A taxonomy of validity evidence: content, criterion, construct, external, consequential (25 min)
– Threats to validity in AI evaluation (20 min)

Lecture 2: Diagnostic Tools

– Differential Item Functioning (DIF) analysis (25 min)
– Dimensionality assessment and parallel analysis (25 min)
– Item-fit statistics and contamination detection (20 min)
– Hands-on: running diagnostics on simulated data (5 min)

Lecture 3: Building Valid Benchmarks

– The Multitrait-Multimethod matrix (25 min)
– Principled item construction and revision (20 min)
– Nomological networks and the claim-centered framework (15 min)
– Design exercise: planning a valid evaluation (15 min)

Notation

This chapter introduces validity-specific notation: g (group membership for DIF), α_{MH} (Mantel-Haenszel odds ratio), λ_k (eigenvalues for dimensionality), MNSQ_i (item fit statistics), and r_{ij} (MTMM correlations). See **?@sec-notation** for the complete notation reference.

6.1 Borsboom's Warrant Inference Framework

Before we can assess whether an evaluation is valid, we must understand what validity *means*. This seemingly philosophical question has profound practical implications. If we do not have a clear conception of validity, we cannot evaluate whether our benchmarks actually measure what we intend.

The Dutch psychometrician Denny Borsboom has developed the most influential contemporary framework for understanding measurement validity. His approach, which we call the *realist framework*, provides the philosophical foundation for the AIMS approach to AI evaluation.

6.1.1 Validity as Truth, Not Evidence

Traditional approaches to validity, following Cronbach and Messick, treat validity as a matter of *evidence accumulation*. Under this view, a test is valid to the extent that we have gathered evidence supporting its intended interpretation. Validity becomes a matter of degree: more evidence means more validity.

Borsboom rejects this view. He argues that validity is fundamentally about *truth*, not evidence:

Borsboom's Definition of Validity

A test is **valid** for measuring an attribute if and only if:

1. The attribute **exists**, and
2. Variations in the attribute **causally produce** variations in the measurement outcomes.

This is a yes/no property: either the attribute causes the test responses, or it does not. Evidence is relevant to our *knowledge* of validity, but validity itself is about the causal structure of the world.

This definition has several important implications:

Existence requirement. The attribute being measured must actually exist. If we claim to measure "general intelligence" but there is no such thing—if intelligence is better understood as a collection of independent abilities—then no test can validly measure it. The existence question is empirical, not definitional.

Causation requirement. The attribute must *cause* variation in test responses. It is not enough for test scores to be *correlated* with the attribute; the attribute must be the reason for the variation. This rules out tests that are merely predictive of outcomes without measuring the underlying construct.

Truth vs. evidence distinction. We can have strong evidence that a test is valid and yet be wrong. Conversely, a test might be valid even if we have limited evidence. This distinction matters because it separates the epistemological question (what do we know?) from the ontological question (what is true?).

6.1.2 The Warrant Inference Problem

Measurement involves an inference from observed data to latent constructs:

$$\text{Observed: } Y_{ij} \quad \xrightarrow{\text{inference}} \quad \text{Latent: } \theta_i$$

This inference requires a **warrant**: a justified belief that the test measures what it claims to measure. The warrant connects the measurement procedure (administering test items, recording responses) to the theoretical construct (ability, intelligence, reasoning).

Following Toulmin's model of argumentation, a measurement argument has the structure:

- **Claim**: "Model i has ability $\theta_i = 2.3$"
- **Data**: "Model i answered 47 of 60 questions correctly"
- **Warrant**: "The test measures the ability construct, and the scoring procedure accurately converts responses to ability estimates"
- **Backing**: "The test items were written by domain experts, the psychometric model fits the data, ability estimates are stable across different item subsets"

The warrant is the critical element. Without it, we have no basis for interpreting test scores as measurements of the intended construct. The backing provides evidence for the warrant but does not replace it.

6.1.3 Semantic Indeterminacy

Borsboom identifies a fundamental problem in measurement: **semantic indeterminacy**. The meaning of a test score depends on which measurement system we adopt, but there is no compelling argument for any particular system.

Consider three measurement frameworks:

1. **Classical Test Theory (CTT)**: A test score $X = T + E$ consists of a true score T plus random error E. The true score is defined as the expected value of the test score over hypothetical replications.

2. **Item Response Theory (IRT)**: Test responses are generated by a latent ability θ through a probabilistic model $P(Y_{ij} = 1|\theta_i, \beta_j)$. The ability parameter is a property of the person that exists independently of any particular test.
3. **Network Models**: There is no latent variable. Test items are causally connected to each other, and correlations arise from these direct connections rather than a common cause.

These frameworks make different claims about what test scores mean:

Framework	What does the score represent?
CTT	Expected value over test replications
IRT	Position on a latent continuum
Network	Summary of a network state

The frameworks are not merely different parameterizations of the same model—they make different ontological commitments about what exists and what causes what. Yet we often cannot empirically distinguish between them.

Implications for AI Evaluation

When we say "GPT-4 has reasoning ability of 2.3 logits," what do we mean? The answer depends on our measurement framework:

- **CTT interpretation**: If we tested GPT-4 many times on parallel forms, its average score would correspond to 2.3 logits.
- **IRT interpretation**: GPT-4 possesses an underlying reasoning capacity that, when combined with item difficulties, generates the observed response pattern.
- **Network interpretation**: GPT-4's responses to reasoning questions form a pattern that we summarize with the number 2.3, but there is no single "reasoning ability" being measured.

These interpretations have different implications for how we should use and trust the measurement.

6.1.4 Construct Validity and the Nomological Network

If a construct cannot be directly observed, how do we know it exists? Cronbach and Meehl proposed that constructs are defined by their place in a **nomological network**—a web of theoretical relationships connecting the construct to other constructs and observable indicators.

For example, "reasoning ability" might be defined by relationships like:

- Higher reasoning ability $\to$ better performance on logic puzzles
- Higher reasoning ability $\to$ better performance on mathematical proofs
- Higher reasoning ability $\to$ correlation with general intelligence
- Higher reasoning ability $\to$ development with education

The construct gains meaning through these relationships. If a test score behaves as the theory predicts—if it correlates with the right things and not with the wrong things—we have evidence that it measures the intended construct.

For AI evaluation, this suggests we need theoretical frameworks that specify:

1. What capabilities should be related to benchmark performance
2. What capabilities should be independent of benchmark performance
3. How capabilities should develop with model scale or training
4. How capabilities should transfer across domains

Without such frameworks, we have benchmark scores without meaning.

6.2 *From Reliability to Validity*

Consider a coding benchmark that produces highly consistent scores: every time you evaluate a model, it gets roughly the same accuracy. The split-half reliability is 0.95, Cronbach's alpha is 0.93, and the G-coefficient from Chapter 5 is excellent. You might conclude that the benchmark is working well.

But suppose you investigate further and discover that the benchmark items were scraped from a popular programming tutorial site. The models with the highest scores turn out to be those whose training data included that site. The benchmark is not measuring *coding ability*—it is measuring *memorization of specific solutions*. The scores are perfectly consistent, but they do not mean what you think they mean.

This is the validity problem. Reliability asks: *does the evaluation give the same answer twice?* Validity asks the deeper question: *does the evaluation measure what it claims to measure?* A perfectly reliable instrument can be completely invalid—like a bathroom scale that always reads 70 kg regardless of who steps on it. Conversely, validity *requires* reliability: if measurements fluctuate randomly, they cannot track any underlying attribute.

In Section 6.1, 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). This is a demanding standard. It requires not just statistical association between scores and the target construct, but a causal mechanism connecting the two.

More recently, Salaudeen et al. (2025) proposed a *claim-centered* framework for AI evaluation validity. Their key insight is that benchmarks are not valid or invalid in the abstract—they are valid *for specific claims*. A benchmark might validly support the claim "Model A scores higher than Model B on these 500 multiple-choice questions" while failing to support the broader claim "Model A reasons better than Model B." Validity is about the relationship between the measurement instrument, the intended interpretation, and the population of systems being evaluated.

! Validity Is Not a Property of the Test Alone

A benchmark is not "valid" or "invalid" in isolation. Validity depends on:

- **The construct**: what attribute are we claiming to measure?
- **The population**: which models or systems are being evaluated?
- **The interpretation**: what conclusions are drawn from the scores?
- **The context**: in what setting will the scores be used?

The same benchmark may be valid for one purpose and invalid for another. Validity is a property of the *inference*, not the instrument.

This chapter develops the theory and tools for assessing and building validity in AI evaluation. We begin with a taxonomy of validity evidence (Section 6.3), then examine the major threats to validity in AI benchmarks (Section 6.4). We develop diagnostic tools for detecting these threats (Section 6.5) and close with methods for constructing valid evaluation instruments (Section 6.6).

6.3 A Taxonomy of Validity Evidence

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.

6.3.1 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 interpretation? 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.

6.3.2 *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"?

6.3.3 *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 6.6.1.

Under Borsboom's causal framework, construct validity requires a specific causal claim: variation in the latent attribute (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).

6.3.4 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 7. 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.

6.3.5 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.

6.3.6 Summary

Table 6.2 summarizes the five forms of validity evidence, their key questions, and how they manifest in AI evaluation.

Table 6.2
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

6.4 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.

6.4.1 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).

6.4.2 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 6.3.1 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.

6.4.3 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.

6.4.4 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:

$$P(X_{ij} = 1 \mid \theta_j, g = 1) \neq P(X_{ij} = 1 \mid \theta_j, g = 0) \tag{6.1}$$

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:

$$\alpha_{MH} = \frac{\sum_k A_k D_k / N_k}{\sum_k B_k C_k / N_k} \tag{6.2}$$

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.

6.5 Diagnostic Tools for Validity

This section develops three computational tools for detecting the validity threats described above.

6.5.1 DIF Analysis

The Mantel-Haenszel procedure described in Section 6.4.4 provides a nonparametric DIF test. A complementary approach uses **logistic regression** (Zumbo 1999):

$$\log \frac{P(X_{ij} = 1)}{P(X_{ij} = 0)} = \beta_0 + \beta_1 \theta_j + \beta_2 g_j + \beta_3 (\theta_j \times g_j) \tag{6.3}$$

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
```

```python
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,
})
```

```python
#| 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).

6.5.2 Dimensionality Assessment

A benchmark that claims to measure a single construct (e.g., "reasoning") should exhibit a unidimensional factor structure: a single latent factor should explain the bulk of the covariance among items. If multiple factors are needed, the benchmark is actually measuring multiple constructs, and a single summary score conflates them.

The primary tool for dimensionality assessment is **eigenvalue analysis** of the item correlation matrix. In a unidimensional benchmark, the first eigenvalue should be substantially larger than the rest. However, raw eigenvalues can be misleading because even random data produces a non-trivial first eigenvalue.

Parallel analysis (Horn 1965) addresses this by comparing the observed eigenvalues to those from random data of the same dimensions. A factor is retained only if its eigenvalue exceeds the corresponding random-data eigenvalue. This is connected to the factor models discussed in Chapter 2: if the data require k factors, the benchmark is measuring k distinct constructs, not one.

```python
#| 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

# Two latent factors with moderate correlation
factor_corr = 0.3
cov_matrix = np.array([[1.0, factor_corr], [factor_corr, 1.0]])
L = np.linalg.cholesky(cov_matrix)
factors = np.random.normal(0, 1, (n_models, 2)) @ L.T

# Factor loadings
```

```
loadings = np.zeros((n_items, 2))
loadings[:n_factor1, 0] = np.random.uniform(0.5, 0.9, n_factor1)  # Factor 1
loadings[n_factor1:, 1] = np.random.uniform(0.5, 0.9, n_factor2)  # Factor 2

# Generate continuous item scores (for eigenvalue analysis)
noise_var = 0.4
X = factors @ loadings.T + np.random.normal(0, np.sqrt(noise_var), (n_models,
 ↪  n_items))

# --- Eigenvalue analysis ---
corr_matrix = np.corrcoef(X.T)
eigenvalues = np.linalg.eigvalsh(corr_matrix)[::-1]  # Descending

# --- Parallel analysis (100 replications) ---
n_reps = 100
random_eigenvalues = np.zeros((n_reps, n_items))
for rep in range(n_reps):
    X_random = np.random.normal(0, 1, (n_models, n_items))
    corr_random = np.corrcoef(X_random.T)
    random_eigenvalues[rep] = np.linalg.eigvalsh(corr_random)[::-1]

pa_threshold = np.percentile(random_eigenvalues, 95, axis=0)

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

# Panel 1: Scree plot with parallel analysis
components = np.arange(1, n_items + 1)
axes[0].plot(components[:10], eigenvalues[:10], 'o-', color='#5B8DEE', linewidth=2,
             markersize=8, label='Observed eigenvalues', zorder=3)
axes[0].plot(components[:10], pa_threshold[:10], 's--', color='#E8637A', linewidth=2,
             markersize=6, label='95th percentile (random)', zorder=3)
axes[0].axhline(y=1, color='gray', linewidth=0.8, linestyle=':', label='Kaiser
 ↪  criterion')
# Shade retained factors
n_retained = sum(eigenvalues > pa_threshold)
for k in range(min(n_retained, 10)):
    axes[0].axvspan(k + 0.6, k + 1.4, alpha=0.1, color='#45BF7C')
axes[0].set_xlabel('Component')
axes[0].set_ylabel('Eigenvalue')
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)
```

```python

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 analysis connects directly to the factor models introduced in Chapter 2. When dimensionality assessment reveals multiple factors, the appropriate response is to either (a) report subscores for each factor rather than a single composite, or (b) revise the benchmark to focus on a single construct.

6.5.3 *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.

```python
#| 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])

axes[1].plot(theta_smooth, icc_expected, '#5B8DEE', linewidth=2, label='Expected
↪  (Rasch)')
axes[1].scatter(theta_mid, obs_prop, color='#E8637A', s=80, zorder=3,
                edgecolors='white', linewidth=0.5, label='Observed proportions')
axes[1].set_xlabel(r'Ability ($\theta$)')
axes[1].set_ylabel('P(Correct)')
axes[1].set_title(f'Item {item_idx + 1} (Contaminated): Observed vs Expected')
axes[1].legend()
axes[1].set_ylim(-0.05, 1.05)

```

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

# Summary
print(f"\nItem Fit Summary:")
print(f"  Items with outfit > 1.3: {sum(1 for x in outfit if x > 1.3)}")
print(f"  Items with infit > 1.3: {sum(1 for x in infit if x > 1.3)}")
contaminated_flagged = sum(1 for i in contaminated_items if outfit[i] > 1.3 or
 ↪  infit[i] > 1.3)
print(f"  Contaminated items flagged: {contaminated_flagged} / {n_contaminated}")
print(f"\n  Contaminated item fit values:")
for i in contaminated_items:
    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.

6.6 Building Valid Benchmarks

The diagnostic tools above detect validity problems in existing benchmarks. This section turns to *constructing* benchmarks with validity built in from the start.

6.6.1 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.

```python
#| 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])
```

```python

# 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()

print(f"\nMTMM Validity Check:")
print(f"  Convergent validity (mean): {np.mean(convergent):.3f}")
print(f"  Discriminant - monomethod (mean): {np.mean(discriminant_mono):.3f}")
print(f"  Discriminant - heteromethod (mean): {np.mean(discriminant_hetero):.3f}")
```

```

# Campbell-Fiske criteria
cf1 = all(c > 0 for c in convergent)
cf2 = np.mean(convergent) > np.mean(discriminant_hetero)
cf3 = np.mean(convergent) > np.mean(discriminant_mono)
print(f"\n  Campbell-Fiske Criteria:")
print(f"    1. Convergent > 0: {'PASS' if cf1 else 'FAIL'}")
print(f"    2. Convergent > heterotrait-heteromethod: {'PASS' if cf2 else 'FAIL'}")
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.

6.6.2 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 6.5.3).

5. **Screen for validity threats**: Run DIF analysis (Section 6.5.1) to check for group bias. Run dimensionality analysis (Section 6.5.2) 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.

```python
#| 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)
```

```python

# 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.

6.6.3 *Nomological Networks*

The concept of the **nomological network** was introduced by Cronbach and Meehl (1955) as the broader theoretical context in which a construct's validity is established. A construct does not exist in isolation—it is defined by its lawful relationships with other constructs, observable behaviors, and experimental outcomes.

For AI evaluation, a nomological network specifies:

- **What the construct should predict**: "Mathematical reasoning" should predict success on novel math problems, performance in tutoring scenarios, and ability to verify proofs.
- **What should predict the construct**: Training on diverse mathematical data, chain-of-thought prompting, and model scale should all increase mathematical reasoning scores.
- **What the construct should be distinct from**: Mathematical reasoning should be separable from memorization of specific problem-solution pairs, linguistic fluency, and pattern matching on surface features.

Salaudeen et al. (2025) operationalize this through their claim-centered framework: each benchmark should be accompanied by explicit claims about what the scores mean, and validity evidence should be gathered for each claim. This is more practical than the full nomological network, which requires specifying all theoretical relationships, but serves the same purpose of grounding scores in a web of meaning.

The MTMM analysis from Section 6.6.1 provides empirical evidence for the nomological network by testing whether the expected convergent and discriminant relationships actually hold. Item-level diagnostics from Section 6.5.1 and Section 6.5.3 ensure that individual items function as predicted by the construct theory.

From Borsboom to Salaudeen: Two Views of Validity

Borsboom's realist framework asks: *does the attribute exist, and does it causally produce score variation?* This is a strong ontological claim that requires causal evidence.

Salaudeen et al.'s claim-centered framework asks: *what specific claims does this benchmark support, and what evidence exists for each?* This is more pragmatic, acknowledging that validity comes in degrees and depends on use.

The two frameworks are complementary. Borsboom's framework sets the philosophical standard; Salaudeen et al.'s framework operationalizes it for practical AI evaluation. Both agree that validity is not a binary property and that multiple forms of evidence are needed.

6.7 *Designing Valid AI Evaluations*

We close with six design principles that synthesize the concepts developed in this chapter.

1. **Start with the construct, not the data**. Define what you are measuring 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 you want 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.

6.8 Discussion Questions

1. 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 you want, and why?
2. How does benchmark contamination differ from construct-irrelevant variance? Can you describe a scenario where training data overlap is actually *construct-relevant* rather than a validity threat?
3. 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.
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 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?

6.9 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 attribute exists and causally produces score variation. This causal perspective connects validity to the structural causal models discussed in Chapter 7.

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). Parallel analysis for dimensionality assessment was proposed by Horn (1965).

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 7.

6.10 Exercises

Theoretical

1. 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?

2. 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?

3. 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?

4. 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?

Computational

5. 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?

6. Simulate a benchmark with 3 underlying factors (e.g., verbal, quantitative, and spatial reasoning) using a factor model from Chapter 2. 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?

7. Using the MTMM simulation framework from Section 6.6.1, 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.

Discussion

8. 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 attribute 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?

7 Causality and Distribution Shift

Intended Learning Outcomes

By the end of this chapter, you 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), $\widehat{V}_{\text{DR}}$ (doubly robust estimator), and $C_\alpha(x)$ (conformal prediction sets). See **?@sec-notation** for the complete notation reference.

7.1 *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 2, 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 6, 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.

7.1.1 *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.

7.1.2 *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
- $V = \{V_1, \dots, V_n\}$ is a set of endogenous variables
- $F = \{f_1, \dots, f_n\}$ 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.

7.1.3 *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 6.4 correspond to specific DAG structures:

– **Contamination** (Section 6.4.3): 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 6.4.1): 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 6.4.4): 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 6.5 — DIF analysis, dimensionality assessment, item-fit statistics — are empirical tests for whether such paths exist.

7.2 *Interventional and Counterfactual Diagnostics*

The evaluation DAG from Section 7.1.3 allows us to reason about *interventions*—what happens when we deliberately change one variable while holding others fixed.

7.2.1 *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:

$$P(Y \mid \mathrm{do}(X = x)) = \sum_z P(Y \mid X = x, Z = z) P(Z = z) \tag{7.1}$$

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.

7.2.2 *Diagnosing Contamination*

The contamination diagnostic from Section 6.4.3 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:

$$P(Y \mid \mathrm{do}(D_{\text{overlap}} = \emptyset)) \overset{?}{=} P(Y \mid D_{\text{overlap}} = \emptyset)$$

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 6.5.3 detect items where some models perform unexpectedly well—a signature of the memorization path.

7.2.3 *Counterfactual Reasoning for DIF*

Differential Item Functioning (Section 6.4.4) 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 6.5.1 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 6.5 are complementary approaches to detecting these unblocked paths.

7.3 *Distribution Shift and Transportability*

In Section 6.3.4, we raised the question of whether benchmark results generalize beyond the specific conditions of the evaluation. This section provides the formal framework.

7.3.1 *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.

7.3.2 *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 6.3.1: a benchmark with poor content validity (non-representative item sampling) is a benchmark with covariate shift relative to the construct domain.

7.3.3 *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.

7.3.4 *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
```

```python
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.

7.4 *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.

7.4.1 *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 4.2.2), 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 $\pi_0 \neq \pi$.

7.4.2 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:

$$\hat{V}_{\text{DM}} = \frac{1}{n} \sum_{t=1}^{n} \hat{r}(x_t, \pi) \tag{7.2}$$

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 2: $\hat{r}(x, a) = P(Y = 1 \mid \theta, \beta_x)$. The direct method is what we have been doing throughout Chapters 1–3: 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:

$$\hat{V}_{\text{IPW}} = \frac{1}{n} \sum_{t=1}^{n} \frac{\pi(a_t \mid x_t)}{\pi_0(a_t \mid x_t)} r_t \tag{7.3}$$

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 4.2.2), 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, Langford, and Li 2011; Robins, Rotnitzky, and Zhao 1994) combines both approaches:

$$\hat{V}_{\mathrm{DR}} = \frac{1}{n}\sum_{t=1}^{n}\left[\hat{r}(x_t,\pi) + \frac{\pi(a_t \mid x_t)}{\pi_0(a_t \mid x_t)}(r_t - \hat{r}(x_t,a_t))\right] \tag{7.4}$$

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 $\hat{V}_{\mathrm{DR}} \approx \hat{V}_{\mathrm{DM}}$.
- 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 $\mathbb{E}[\hat{V}_{\mathrm{DR}}] = V(\pi)$ when either model is correct. Taking expectations:

$$\mathbb{E}[\hat{V}_{\mathrm{DR}}] = \mathbb{E}[\hat{r}(x,\pi)] + \mathbb{E}\left[\frac{\pi(a \mid x)}{\pi_0(a \mid x)}(r - \hat{r}(x,a))\right]$$

For the second term, conditioning on x:

$$\begin{aligned}\mathbb{E}\left[\frac{\pi(a \mid x)}{\pi_0(a \mid x)}(r - \hat{r}(x,a)) \,\middle|\, x\right] &= \sum_a \pi_0(a \mid x)\frac{\pi(a \mid x)}{\pi_0(a \mid x)}(\mathbb{E}[r \mid x,a] - \hat{r}(x,a)) \\ &= \sum_a \pi(a \mid x)(r^*(x,a) - \hat{r}(x,a))\end{aligned}$$

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:

$$\mathbb{E}[\hat{V}_{\mathrm{DR}}] = \mathbb{E}[\hat{r}(x,\pi)] + \mathbb{E}_x\left[\sum_a \pi(a \mid x)(r^*(x,a) - \hat{r}(x,a))\right] = \mathbb{E}_x\left[\sum_a \pi(a \mid x) r^*(x,a)\right] = V(\pi)$$

7.4.3 *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 $\mu = \mathbb{E}[Y]$:

$$\hat{\mu}_{\text{PPI}} = \frac{1}{N}\sum_{i\in\mathcal{D}_N}\hat{Y}_i + \frac{1}{n}\sum_{i\in\mathcal{D}_n}(Y_i - \hat{Y}_i) \tag{7.5}$$

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 2.6 is direct. The PPE (Prediction-Powered Evaluation) approach from Chapter 2 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 $Y - \hat{Y}$.

7.4.4 *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):

$$\hat{\mu}_{\text{IW}} = \frac{\sum_{i=1}^n w(x_i)Y_i}{\sum_{i=1}^n w(x_i)}, \quad w(x_i) = \frac{P^{(t)}(x_i)}{P^{(s)}(x_i)} \tag{7.6}$$

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, Suzuki, and Kanamori 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.

```python
#| 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.

7.5 Conformal Prediction Under Distribution Shift

The off-policy estimators in Section 7.4 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.

7.5.1 Split Conformal Prediction

The split conformal algorithm (Vovk, Gammerman, and Shafer 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 $\{s_1, \dots, s_n\}$.
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.

7.5.2 *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**.

7.5.3 *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 7.4.4: reweight the calibration scores to match the target distribution.

The algorithm modifies step 3: instead of the uniform quantile, compute the weighted quantile of $\{s_1, \dots, s_n, +\infty\}$ with weights $\{w_1, \dots, w_n, w_{n+1}\}$ 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.

```python
#| 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)

# Test at multiple nominal coverage levels
alphas = np.arange(0.05, 0.55, 0.05)
coverage_naive = np.zeros(len(alphas))
coverage_weighted = np.zeros(len(alphas))

for a_idx, alpha in enumerate(alphas):
    covers_naive = 0
    covers_weighted = 0
    n_total = 0

    for j in range(n_models):
        theta_j = np.random.normal(0, 1)

        # True accuracy on calibration and test items
        p_cal = expit(theta_j - beta_cal)
        p_test = expit(theta_j - beta_test)

        # Generate binary responses
        y_cal = np.random.binomial(1, p_cal)
        y_test = np.random.binomial(1, p_test)

        # Model prediction: IRT probability
        r_hat_cal = p_cal
        r_hat_test = p_test

        # Nonconformity scores on calibration set
        scores_cal = np.abs(y_cal - r_hat_cal)

        # --- Naive conformal: uniform quantile ---
        q_level = int(np.ceil((1 - alpha) * (n_cal + 1))) - 1
        q_level = min(q_level, n_cal - 1)
        sorted_scores = np.sort(scores_cal)
        q_naive = sorted_scores[q_level]

        # --- Weighted conformal: importance-weighted quantile ---
        weights = norm.pdf(beta_cal, 1.0, 1.0) / norm.pdf(beta_cal, -0.5, 0.8)
        weights = weights / weights.sum()

        # Weighted quantile
        sorted_idx = np.argsort(scores_cal)
        cum_weights = np.cumsum(weights[sorted_idx])
        q_idx = np.searchsorted(cum_weights, 1 - alpha)
        q_idx = min(q_idx, n_cal - 1)
        q_weighted = scores_cal[sorted_idx[q_idx]]

        # Check coverage on test items
        for i in range(n_test):
            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()

```

```python
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).

7.6 *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 7.1.3). 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 7.3). 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 7.4.2). 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 7.5.3). 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 7.2.2). 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 6.5.3) 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 6.5, providing a systematic approach to evaluating whether benchmark results are trustworthy enough to inform deployment decisions.

7.7 *Discussion Questions*

1. A benchmark uses Computerized Adaptive Testing (Section 4.2.2) 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 you would correct the estimate 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 would you detect the failure?

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?

7.8 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, Janzing, and Schölkopf (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, Suzuki, and Kanamori (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, Rotnitzky, and Zhao (1994), who introduced the augmented inverse probability weighted estimator. Dudı□k, Langford, and Li (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, Gammerman, and Shafer (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 2. The evaluation DAG in this chapter makes that connection explicit, showing how each validity threat from Section 6.4 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.

7.9 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 $\hat{\mu}_{\mathrm{IW}} = \frac{1}{n}\sum_{i=1}^{n} w(x_i)Y_i$ 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 $\hat{\mu}_{\mathrm{IW}}$ diverge?

2. Prove the double robustness property: show that $\mathbb{E}[\hat{V}_{\mathrm{DR}}] = V(\pi)$ (Equation 7.4) 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 7.1.3, 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 4.2.2. 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 2.6. Design a practical PPI pipeline for evaluating a new model on 10,000 items when you can only afford to run 500 actual evaluations. 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?

III

Design, Governance, and Applications

8 Information and Mechanism Design

Intended Learning Outcomes

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

1. **Explain** why Goodhart's Law is not merely a cautionary aphorism but a formal game-theoretic phenomenon, and classify its four variants in AI evaluation contexts.
2. **Formalize** AI evaluation as a Stackelberg game between an evaluation designer and a model builder, and prove that deterministic benchmarks fail while randomized mechanisms achieve one-shot incentive alignment.
3. **Analyze** the information-variance tradeoff in repeated evaluation: why reducing noise requires revealing information, and how this degrades alignment over time.
4. **Derive** distribution correction as the primary mechanism for restoring alignment, and compute the optimal evaluation size k^* as a function of correction rate ρ.
5. **Model** metric design as a principal-agent problem and characterize when developers prefer to reveal, conceal, or garble evaluation information.
6. **Apply** positional representation and positional proportionality criteria to select representative benchmark subsets using social choice theory.
7. **Synthesize** design principles spanning strategic robustness, information control, and representativeness for AI benchmarks.

Suggested Lecture Plan

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

Lecture 1: The Evaluation Game

- Goodhart's Law and strategic manipulation (15 min)
- Stackelberg benchmark game: setup and failure of deterministic mechanisms (20 min)
- One-shot incentive alignment via randomization (25 min)
- Hands-on: deterministic vs. randomized evaluation simulation (15 min)

Lecture 2: Information Leakage and Restoring Alignment

- Repeated evaluation and the information-variance tradeoff (20 min)
- Distribution correction and alignment recovery (25 min)
- Optimal evaluation size and the holdout mechanism (20 min)
- Hands-on: Pareto frontier and correction simulations (10 min)

Lecture 3: Metric Design and Representative Selection

- Metric design as a principal-agent problem (25 min)

- Information elicitation: reveal, conceal, garble (20 min)
- Representative benchmark selection via social choice theory (20 min)
- Design principles synthesis (10 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), Δ_t (misalignment), and γ (gaming penalty). See **?@sec-notation** for the complete notation reference.

8.1 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 2), how to quantify measurement precision (Chapter 5), and how to assess whether a benchmark measures what it claims to measure (Chapter 6, Chapter 7). 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 6.4.3).

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 Chapters 3–6 to this chapter is a shift in the source of the threat. Measurement error (Chapter 5) is stochastic. Validity threats (Section 6.4) 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.

The remainder of this chapter builds three layers of strategic analysis: (i) benchmark disclosure and information design — how much should evaluators reveal about the evaluation mechanism? (ii) metric selection and reporting granularity — which metrics should evaluators report, and at what level of detail? (iii) mechanism design for repeated evaluation — how can evaluators maintain alignment when the game repeats?

8.2 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?

8.2.1 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 the aggregate performance across all tasks:

$$u_E(\theta) = \sum_{f \in F} f(\theta)$$

The designer acts as a social planner seeking models that perform broadly rather than narrowly.

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.

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.

8.2.2 *Failure of Deterministic Mechanisms*

Consider first a deterministic mechanism where the designer publishes a fixed subset $S^* \subseteq F_E$.

Proposition: Failure of Deterministic Mechanisms

If the designer publishes a deterministic set S^* with additive reward $r(\theta, S) = \sum_{f \in S} f(\theta)$, the builder's best response is:

$$\theta^* \in \arg\max_{\theta \in \Theta} \sum_{f \in S^*} f(\theta)$$

This provides *no incentive* for performance on tasks outside S^*.

Proof. The builder observes S^* directly and maximizes the known objective. Tasks in $F \setminus S^*$ do not affect the score. □

This is exactly Goodhart's adversarial variant: the builder exploits the known structure of the evaluation to specialize. It is also the construct underrepresentation problem from Section 6.4.2 viewed through a strategic lens: a fixed benchmark *systematically ignores* capabilities outside S^*, and a rational builder responds accordingly.

8.2.3 *One-Shot Alignment via Randomization*

While deterministic mechanisms fail, randomization offers a path forward. When the builder faces genuine uncertainty about which tasks will be evaluated, the strategic landscape changes fundamentally.

Theorem 1: One-Shot Incentive Alignment (Omniscient Builder)

Consider a randomized mechanism where (i) M draws a single task uniformly: $s \sim \text{Unif}(F_E)$, (ii) the reward is $r(\theta, \{s\}) = s(\theta)$, and (iii) the builder has a uniform prior π over which tasks comprise F_E. Then the builder's expected reward is proportional to the designer's utility:

$$\mathbb{E}_{F_E \sim \pi}\left[\mathbb{E}_{s \sim \text{Unif}(F_E)}[s(\theta)]\right] = \frac{1}{|F|}\, u_E(\theta)$$

and the builder's best response maximizes $u_E(\theta)$.

Proof sketch. Under a symmetric (uniform) prior over which tasks compose F_E, the builder has no information distinguishing any task f_i from f_j. By symmetry, the marginal probability that any specific task f is the sampled task is constant: $P(s = f) = 1/|F|$. Therefore the builder's expected reward is $(1/|F|) \sum_{f \in F} f(\theta) = (1/|F|)\, u_E(\theta)$, and maximizing expected reward is equivalent to maximizing $u_E(\theta)$. □

The omniscient version assumes the builder knows all of F. A more realistic version considers a builder with limited information:

Theorem 2: One-Shot Alignment (Limited Information)

Suppose the builder draws tasks i.i.d. from distribution p_M and believes the evaluator also draws from p_M. Under a single-sample mechanism, the builder's best response is:

$$\theta^* \in \arg\max_{\theta \in \Theta} \mathbb{E}_{f \sim p_M}[f(\theta)]$$

That is, the builder optimizes over their own best approximation of the task universe.

Proof sketch. The builder's expected reward, taken over their prior on F_E and the randomness in M, equals $\mathbb{E}_{f \sim p_M}[f(\theta)]$. Since F_M is drawn from p_M, the builder's best response is empirical risk minimization over F_M, which is the best they can do given their information. □

This result is powerful: the single-sample mechanism is "the best you can do" as an evaluation designer. The builder's best response is to approximate their own approximation of F to the best of their ability — essentially training broadly over all available data. Under incentive alignment, the builder performs **empirical risk minimization** (ERM) over their task set F_M. By classical uniform convergence (Vapnik 1998), restricting to any subset $S \subset F_M$ can only increase regret; the builder has no incentive to "game" by focusing on a narrow task set.

8.2.4 *Discussion: What Randomization Buys*

The one-shot alignment result reveals a fundamental tension with the statistical efficiency goals of earlier chapters. In **?@sec-fisher-information** and Section 4.2.2, 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 4.2.2) 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,
})
```

```python
#| 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

# Task values: how much each task contributes to "true capability"
task_values = np.random.uniform(0.3, 1.0, N)

# --- Deterministic mechanism: builder knows 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 = sum of performance across ALL tasks
det_utility = det_perf.sum()
rand_utility = rand_perf.sum()

# 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('Deterministic')
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 / N, rand_utility / N]
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(['Det.', 'Rand.'])
ax.set_ylabel('Avg. true utility')
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 deterministic mechanism (left), the builder concentrates effort on the $k = 10$ known evaluation tasks and neglects the rest, achieving high benchmark scores but low true utility. Under a randomized mechanism (center), the builder spreads effort broadly. The right panel compares average true utility across *all* tasks: the randomized mechanism produces a substantially higher utility despite lower peak performance on any single task.

8.3 The Information-Variance Tradeoff

Theorem 1 established that randomized evaluation achieves incentive alignment in one-shot settings. But benchmarks operate as *repeated games*: weekly leaderboards, monthly submissions, quarterly benchmark releases. In repeated evaluation, randomization alone fails because information *leaks*.

8.3.1 *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.

The variance of a k-task average reward scales as $\mathrm{Var}(\hat{r}_k(\theta)) \approx \sigma^2/k$. 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 $L_t = |\mathcal{I}_t|/|F|$.

8.3.2 *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.

8.3.3 The Pareto Frontier

Define the **residual misalignment** at time t as:

$$\Delta_t = \mathbb{E}_{f\sim\pi_E^{(t)}}[f(\theta_t^*)] - \frac{1}{|F|}u_E(\theta_t^*)$$

This measures the gap between what the builder optimizes for (expected performance under π_E) and what society wants (aggregate performance over all of F). When π_E is uniform, $\Delta_t = 0$.

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

# Initial biased evaluator distribution: overweight first 40% of tasks by 3x
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: TV distance as proxy
            tv = 0.5 * np.sum(np.abs(pi_hat - np.ones(N)/N))
            bias_tv = 0.5 * np.sum(np.abs(pi_E_true - np.ones(N)/N))
            # Builder exploits learned bias: misalignment ~ min(learned_bias,
↪   true_bias)
            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")
```

```python

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.

8.4 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.

8.4.1 Distribution Correction

Assumption: Distribution Correction

The evaluator updates $\pi_E^{(t)}$ via linear interpolation toward uniform at rate $\rho \in (0, 1]$:

$$\pi_E^{(t)} = (1 - \rho)\,\pi_E^{(t-1)} + \rho \cdot \text{Uniform}(F)$$

so that $\text{KL}(\pi_E^{(t)} \| \text{Uniform}(F)) \to 0$ exponentially with rate ρ.

In practice, evaluators correct their biases through several channels: *incident reports* from users who discover model failures that benchmarks missed; *systematic audits* of coverage across capability dimensions (languages, modalities, reasoning types); *improving benchmark acquisition capacity* over time. The parameter ρ captures the fraction of bias removed per evaluation round.

Proposition: Alignment Recovery Under Distribution Correction

Under the repeated evaluation game with distribution correction, the builder's optimal strategy converges to maximizing the designer's utility:

$$\theta_t^* \to \arg\max_{\theta\in\Theta} \frac{1}{|F|}\sum_{f\in F} f(\theta) = \arg\max_{\theta\in\Theta} u_E(\theta)$$

as $\pi_E^{(t)} \to \text{Uniform}(F)$.

Proof sketch. The builder's expected reward under mechanism M is $\mathbb{E}_{f\sim\pi_E^{(t)}}[f(\theta)]$. As $\pi_E^{(t)} \to \text{Uniform}(F)$, this converges to $(1/|F|)u_E(\theta)$. In the limit, the builder knows $\pi_E = \text{Uniform}(F)$ perfectly, but this knowledge provides no advantage: optimizing for the uniform distribution *is* the designer's objective. □

The key insight is a *race between two learners*: the builder learns $\pi_E^{(t)}$ from observations, while the evaluator corrects $\pi_E^{(t)}$ toward uniform. The residual misalignment depends on the relative rates of these two processes.

Proposition: Misalignment Bound

Let $D_0 = \text{KL}(\pi_E^{(0)} \| \text{Uniform}(F))$ and $m_t = k \cdot \min(t, \rho^{-1})$ be the builder's effective sample size. The residual misalignment satisfies:

$$\Delta_t \leq \min\left(\underbrace{\frac{m_t}{m_t + |F|}}_{\text{estimation-limited}}, \underbrace{(1-\rho)^t\sqrt{\frac{D_0}{2}}}_{\text{correction-limited}} \right)$$

The two terms capture complementary regimes:

– **Estimation-limited** (t small): $m_t \approx kt$, giving $\Delta_t \lesssim kt/|F|$. The builder has too few observations to identify the evaluator's biases.
– **Correction-limited** (t large): $\Delta_t \lesssim (1-\rho)^t\sqrt{D_0/2}$. The builder may know $\pi_E^{(t)}$ well, but the evaluator's correction has driven the bias toward zero, leaving nothing to exploit.

Proof sketch. The correction-limited term follows from Pinsker's inequality: $\Delta_t \leq \text{TV}(\pi_E^{(t)}, \text{Uniform}(F)) \leq \sqrt{D_0/2}\,(1-\rho)^t$, since the total variation decays geometrically under linear interpolation. The estimation-limited term follows from Bayesian shrinkage: a builder with m_t observations and a Dirichlet$(1, \ldots, 1)$ prior has posterior mean that is an $m_t/(m_t + |F|)$-fraction of the way from uniform to the empirical distribution. □

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

np.random.seed(42)
```

```

# --- Parameters ---
N = 200
n_rounds = 200
k = 20
rho_values = [0, 0.01, 0.05, 0.1, 0.2]
n_trials = 20

# Initial biased evaluator distribution
pi_E_init = np.ones(N) / N
pi_E_init[:int(0.4 * N)] *= 3.0
pi_E_init /= pi_E_init.sum()

# True task performance: each task has a "value"
task_values = np.random.uniform(0.3, 0.9, N)

colors = ['#888888', '#B07CD8', '#F0A35C', '#45BF7C', '#5B8DEE']

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

for ri, rho in enumerate(rho_values):
    all_misalign = 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()
        uniform = np.ones(N) / N

        for t in range(n_rounds):
            # Correct distribution
            if rho > 0:
                pi_E = (1 - rho) * pi_E + rho * uniform

            # Misalignment: total variation between pi_E and uniform
            tv = 0.5 * np.sum(np.abs(pi_E - uniform))
            all_misalign[trial, t] = tv

            # KL divergence
            mask = pi_E > 1e-15
            kl = np.sum(pi_E[mask] * np.log(pi_E[mask] / uniform[mask]))
            all_kl[trial, t] = kl

    col = colors[ri]
    rounds = np.arange(n_rounds)
    label = f' ={rho}' if rho > 0 else ' =0 (no correction)'

    axes[0].plot(rounds, all_misalign.mean(axis=0), color=col, label=label, lw=1.5)
    kl_mean = all_kl.mean(axis=0)
    kl_mean = np.maximum(kl_mean, 1e-10)
    axes[1].plot(rounds, kl_mean, color=col, label=label, lw=1.5)

axes[0].set_xlabel('Round')
axes[0].set_ylabel('Misalignment (TV distance)')
axes[0].set_title('Residual misalignment')
axes[0].legend()

axes[1].set_xlabel('Round')
axes[1].set_ylabel('KL divergence')
axes[1].set_title('Distribution bias')
axes[1].set_yscale('log')
axes[1].legend()

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

Figure. Alignment recovery under distribution correction with $|F| = 200$ and $k = 20$. Left: residual misalignment converges to zero for all $\rho > 0$, with faster correction rates producing faster convergence. The $\rho = 0$ baseline (gray) maintains persistent misalignment. Right: KL divergence decays exponentially on a log scale, with rate proportional to ρ.

8.4.2 Optimal Evaluation Size

With distribution correction, the Pareto frontier from Section 8.3.3 collapses. Each revealed task provides only *transient* advantage to the builder, because the bias it reveals gets corrected within $O(1/\rho)$ rounds. The evaluator's per-round loss decomposes as:

$$\mathcal{L}(k) = \underbrace{\frac{\sigma^2}{k}}_{\text{evaluation variance}} + \underbrace{\frac{\gamma k}{\rho}}_{\text{steady-state exploitation}}$$

The first term decreases in k (more tasks, less noise); the second increases in k (more tasks, faster builder learning) and decreases in ρ (faster correction, shorter exploitation window).

Proposition: Optimal Sample Size

Given correction rate $\rho > 0$ and gaming penalty $\gamma > 0$, the optimal number of evaluation tasks per round is:

$$k^* = \sigma\sqrt{\frac{\rho}{\gamma}}$$

with minimum loss $\mathcal{L}(k^*) = 2\sigma\sqrt{\gamma/\rho}$.

Proof. Setting $\partial\mathcal{L}/\partial k = -\sigma^2/k^2 + \gamma/\rho = 0$ gives $k^2 = \sigma^2\rho/\gamma$, hence $k^* = \sigma\sqrt{\rho/\gamma}$. The second-order condition $\partial^2\mathcal{L}/\partial k^2 = 2\sigma^2/k^3 > 0$ confirms this is a minimum. □

Several key implications follow:

- **k^* increases with $\sqrt{\rho}$**: Faster correction allows larger evaluation sets. An evaluator who corrects twice as fast can evaluate $\sqrt{2} \approx 1.4$ times as many tasks.
- **k^* increases with σ**: Higher task variance demands more samples for a reliable signal, just as in the static case.
- **Limiting behavior**: As $\rho \to 0$ (no correction), $k^* \to 0$ — the evaluator should reveal as few tasks as possible. As $\rho \to \infty$ (instant correction), $k^* \to \infty$ — leakage is immediately outdated.
- **The key lever is ρ**: Since $\mathcal{L}(k^*) = 2\sigma\sqrt{\gamma/\rho}$, the evaluator's *optimal* loss decreases as $1/\sqrt{\rho}$. Investing in distribution correction (coverage audits, benchmark changelogs, incident-driven task additions) is the most effective lever for benchmark design.

```python
#| label: optimal-k-surface
#| autorun: true
import numpy as np
import matplotlib.pyplot as plt

sigma = 5.0
gamma = 0.05
rho_values = [0.05, 0.1, 0.2, 0.5]

k_range = np.linspace(1, 80, 200)
colors = ['#E8637A', '#F0A35C', '#45BF7C', '#5B8DEE']

fig, ax = plt.subplots(1, 1, figsize=(3.5, 3))

for i, rho in enumerate(rho_values):
    loss = sigma**2 / k_range + gamma * k_range / rho
    k_star = sigma * np.sqrt(rho / gamma)
    loss_star = 2 * sigma * np.sqrt(gamma / rho)

    ax.plot(k_range, loss, color=colors[i], lw=1.5, label=f' ={rho}')
    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=(8, 5), fontsize=6,
    color=colors[i])

# Static case (rho -> 0): only variance term
ax.plot(k_range, sigma**2 / k_range, '--', color='#888888', lw=1, label=' →0 (static)')

ax.set_xlabel('Evaluation tasks per round (k)')
ax.set_ylabel('Evaluator loss L(k)')
ax.set_title('Optimal evaluation size')
ax.legend()
ax.set_ylim(0, 15)
ax.set_xlim(0, 80)

```

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

Figure. Evaluator loss $\mathcal{L}(k) = \sigma^2/k + \gamma k/\rho$ for different correction rates, with stars marking the optimal k^*. Faster correction (larger ρ) shifts k^* rightward and lowers the minimum loss. The static case ($\rho \to 0$, dashed) is monotonically decreasing, reflecting the irresolvable tradeoff without correction. Parameters: $\sigma = 5$, $\gamma = 0.05$.

8.4.3 Noise-Gated Holdout

Distribution correction makes leaked information obsolete by moving the target. A complementary mechanism *prevents* leakage in the first place by gating the information flow.

Definition: Holdout Evaluation Mechanism

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 $\pi_E^{(t)}$ 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, buying the evaluator substantially more time for distribution correction.

Why Noise Alone Cannot Replace Distribution Correction

The holdout mechanism slows the builder's learning but does not change their *incentives* conditional on what they have learned. Regardless of how much Laplace noise is added, the builder's optimal strategy given their posterior $\hat{\pi}_{E,t}$ remains $\theta_t^* = \arg\max_\theta \mathbb{E}_{f\sim\hat{\pi}_{E,t}}[f(\theta)]$. If π_E is biased and fixed, the builder will eventually learn this bias and specialize accordingly. Only distribution correction, which drives $\pi_E^{(t)} \to \text{Uniform}(F)$, ensures that the builder's eventual knowledge provides no exploitable advantage. This is the key difference from pure differential privacy: DP slows learning; distribution correction eliminates the *incentive* to exploit what is learned.

This connects to the differential privacy literature (Dwork et al. 2015; Dwork and Roth 2014). The holdout mechanism inherits DP guarantees from the Sparse Vector Algorithm, bounding the max-information between the builder's observations and the holdout set. However, DP addresses a different threat: it prevents reconstruction of which specific tasks are in the holdout (overfitting to the test *set*), while our concern is that the builder learns the evaluation *distribution* and specializes accordingly.

8.5 Metric Design as Principal-Agent Problem

We now shift from "which tasks to show" to "which metrics to report." Even with a perfectly designed evaluation mechanism, the *choice of metric* can create perverse incentives.

8.5.1 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 7: counterfactual metrics require causal reasoning about what performance *would have been* absent the developer's effort.

8.5.2 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.

8.5.3 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 8.4.3: 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.

```python
#| 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()
```

```python

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.

8.6 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 $\rightarrow$ BIG-bench Lite, 24), HELM ($\rightarrow$ HELM Lite), and Cal Hospital Compare (hundreds of quality measures $\rightarrow$ 12).

8.6.1 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.

8.6.2 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.

8.6.3 *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 6.5.2 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.

```python
#| 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.

8.7 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 8.1
Six design principles for strategic AI evaluation

Principle	Formal Basis	Recommendation
1. Randomize and Refresh	Theorem 1 (one-shot alignment), Prop. (deterministic failure)	Use randomized evaluation with task renewal. Static benchmarks are Goodhart-vulnerable by construction.
2. Correct and Grow	Prop. (alignment recovery), Prop. (optimal k^*)	Invest in distribution correction (ρ) as the primary lever. The optimal k^* scales 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).

8.8 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 6? Is there a formal connection between σ^2/k (evaluation variance) and Cronbach's α?
3. Can you design a mechanism where the builder's incentive is to improve on the *hardest* tasks rather than the average? How would you modify $u_E(\theta)$ and $r(\theta, S)$?
4. In what sense is the Chatbot Arena a randomized evaluation mechanism? Does it satisfy the conditions of Theorem 1? What information leaks through the adaptive matching process?
5. How should a government regulator set γ (the gaming penalty) for a safety benchmark? What factors should influence this 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?

8.9 Bibliographic Notes

The Stackelberg evaluation game and the information-variance tradeoff are developed in Son Truong et al. (2025). The distribution correction mechanism and optimal evaluation size results are from the same work. 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, Schulman, and Hilton (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.

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.

8.10 Exercises

1. **(Easy)** Show that if $\pi_E = \text{Uniform}(F)$, the one-shot alignment theorem holds for any sample size k, not just $k = 1$. *Hint:* Show that for any k-subset mechanism, the expected reward is proportional to $u_E(\theta)$.

2. **(Easy)** Verify that $k^* = \sigma\sqrt{\rho/\gamma}$ minimizes $\mathcal{L}(k) = \sigma^2/k + \gamma k/\rho$. What is $\mathcal{L}(k^*)$? What happens to k^* when γ doubles?

3. **(Medium)** Derive the estimation-limited term in the misalignment bound using a Dirichlet$(1, \dots, 1)$ prior over distributions on F with m effective observations. Show

that the posterior predictive is the shrinkage estimator $\hat{\pi}_E = \frac{m}{m+N}\hat{p} + \frac{N}{m+N}\text{Uniform}(F)$ and that $\text{TV}(\hat{\pi}_E, \text{Uniform}(F)) \leq m/(m+N)$.

4. **(Medium)** In the agency game with binary X, suppose $C \mid X = 0 \sim \text{Exp}(\lambda_0)$ 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)** Extend the one-shot alignment theorem to the case where the builder has a non-uniform prior p_M over F. Show that the builder's best response maximizes $\mathbb{E}_{f \sim p_M}[f(\theta)]$. Under what conditions does this coincide with maximizing $u_E(\theta)$?
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 Red-Teaming and Adversarial Evaluation

Intended Learning Outcomes

By the end of this chapter, you 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 4.2.2.
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: $\theta_j^{(\mathrm{adv})}/\theta_j^{(\mathrm{std})}$ (adversarial/standard ability), $\alpha_{s,\mathcal{D}}$ (attack success probability), J/s (operational judge / oracle criterion), K (repeated samples), and $\hat{\mu}_{\mathrm{PPI}}$ (prediction-powered inference estimator). See **?@sec-notation** for the complete notation reference.

9.1 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.

9.1.1 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 6.3.1:

- **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.

9.1.2 *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:

$$P(\text{resist}_j \mid \beta_i^{(\text{atk})}, \alpha_i, \theta_j^{(\text{adv})}) = \frac{1}{1 + \exp(-\alpha_i(\theta_j^{(\text{adv})} - \beta_i^{(\text{atk})}))} \tag{9.1}$$

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 6.3.1) 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.

9.1.3 *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 6.4.1): 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.

9.1.4 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 4.2.2).

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.

9.2 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.

9.2.1 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.

9.2.2 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 Sang Truong et al. (2025) for a formal treatment integrating IRT into scaling law estimation.

9.2.3 *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):

$$\mathbb{E}[\text{ASR}_J] = \text{TPR} \cdot \alpha + \text{FPR} \cdot (1 - \alpha) \tag{9.2}$$

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 6.4.4) 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.

9.2.4 *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 6.3.1: 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.

9.3 *Adversarial Robustness Evaluation*

9.3.1 *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 $\theta_j^{(\text{adv})}$:

$$P(X_{ij} = 1) = \frac{1}{1 + \exp(-(\alpha_i^{(\text{std})}\theta_j^{(\text{std})} + \alpha_i^{(\text{adv})}\theta_j^{(\text{adv})} - \beta_i))} \tag{9.3}$$

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.

9.3.2 *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.

9.3.3 *Adaptive Adversarial Testing*

The connection to computerized adaptive testing (Section 4.2.2) 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 4.2.2 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.

```python
#| 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 ($\theta^{(\mathrm{adv})}$) is only weakly correlated with standard accuracy ($\theta^{(\mathrm{std})}$)—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 $\theta^{(\mathrm{std})}$) and adversarial items (loading primarily on $\theta^{(\mathrm{adv})}$), 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.

9.4 Synthetic Data for Evaluation

9.4.1 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?*

9.4.2 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 6.4. Distribution mismatch is a form of covariate shift (Section 7.3). Mode collapse reduces content validity (Section 6.3.1). Evaluation circularity introduces construct-irrelevant variance (Section 6.4.1).

9.4.3 *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:

$$\hat{\beta}_i^{(\text{syn})} \approx \hat{\beta}_i^{(\text{human})} + c \tag{9.4}$$

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
```

```python
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.

9.5 Evaluation at Scale

9.5.1 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.

9.5.2 Prediction-Powered Inference

Prediction-powered inference (PPI), introduced by Angelopoulos et al. (2023) and connected to the off-policy evaluation framework in Section 7.4.3, 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:

$$\hat{\mu}_{\text{PPI}} = \frac{1}{N}\sum_{i=1}^{N}\hat{Y}_i + \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i) \tag{9.5}$$

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.

9.5.3 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 4.2.2: 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.

```python
#| 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]
```

```

# --- Estimators ---
# 1. Synthetic only (biased)
mu_syn = Y_syn.mean()

# 2. Human only (unbiased but high variance)
mu_human = Y_human.mean()

# 3. PPI (combines both)
mu_ppi = mu_syn + (Y_human - Y_syn_human).mean()

# --- Confidence intervals ---
n_bootstrap = 2000
boot_syn = np.zeros(n_bootstrap)
boot_human = np.zeros(n_bootstrap)
boot_ppi = np.zeros(n_bootstrap)

for b in range(n_bootstrap):
    # Bootstrap synthetic
    idx_s = np.random.choice(N_total, N_total, replace=True)
    boot_syn[b] = Y_syn[idx_s].mean()

    # Bootstrap human
    idx_h = np.random.choice(n_human, n_human, replace=True)
    boot_human[b] = Y_human[idx_h].mean()

    # Bootstrap PPI
    idx_s2 = np.random.choice(N_total, N_total, replace=True)
    idx_h2 = np.random.choice(n_human, n_human, replace=True)
    boot_ppi[b] = Y_syn[idx_s2].mean() + (Y_human[idx_h2] -
 ↪  Y_syn_human[idx_h2]).mean()

ci_syn = np.percentile(boot_syn, [2.5, 97.5])
ci_human = np.percentile(boot_human, [2.5, 97.5])
ci_ppi = np.percentile(boot_ppi, [2.5, 97.5])

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

# Panel 1: Point estimates and CIs
methods = ['Synthetic\n(N=2000)', 'Human\n(n=80)', 'PPI\n(N+n)']
estimates = [mu_syn, mu_human, mu_ppi]
cis_low = [ci_syn[0], ci_human[0], ci_ppi[0]]
cis_high = [ci_syn[1], ci_human[1], ci_ppi[1]]
colors = ['#E8637A', '#5B8DEE', '#45BF7C']

for i, (m, est, lo, hi, col) in enumerate(zip(methods, estimates, cis_low, cis_high,
 ↪  colors)):
    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.

9.6 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 9.1) that defines the attack taxonomy and coverage requirements. Generate candidate items through a combination of expert red-teamers (depth) and LLM-based generation (Section 9.4) (breadth). Calibrate item parameters by administering the full bank to a reference panel of models, fitting a multidimensional IRT model (Section 9.3) to estimate item difficulties and discriminations.

Stage 2: Adversarial Adaptive Testing. For each new model, run computerized adaptive testing (Section 4.2.2) 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 $\hat{\theta}_j^{(\text{adv})}$ 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 9.4.3) 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 9.5.2) 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 7.5) 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 7.4.4) to maintain coverage guarantees.

This pipeline connects to the causal audit framework from Section 7.6: 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 9.1) produces items; IRT (Section 9.1.2) calibrates them; CAT (Section 4.2.2) administers them efficiently; synthetic data (Section 9.4) scales them; PPI (Section 9.5.2) corrects for bias; conformal prediction (Section 7.5) quantifies uncertainty. Each tool solves a specific problem. The pipeline composes them into a system that is greater than the sum of its parts.

9.7 *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 your 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 your 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 would you modify the PPI framework?

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 9.2, 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 $\sigma(\theta - \beta_i)$?

9.8 *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 6. 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 9.3 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. Sang Truong 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.

9.9 Exercises

Exercise 1 (Red-team test specification). Write a test specification for red-teaming a medical chatbot. Your 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 9.3 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 9.4.3), 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 you can 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 9.6 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.

10 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.

10.1 Our Approach

10.1.1 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 2 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.

10.1.2 From Models to Methods

Chapter 3 showed how to estimate latent parameters from evaluation data via maximum likelihood, EM algorithms, and Bayesian inference. Chapter 4 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.

10.1.3 *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 6 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 7 connected these ideas to formal causal reasoning, showing when benchmark results generalize across contexts and how to correct for distribution shift.

10.1.4 *Design, Strategy, and Adversarial Evaluation*

Part III moved from the science of measurement to its strategic and practical dimensions. Chapter 8 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 9 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.

10.1.5 *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 6). Which models we test and how we sample them shapes the ability scale (Chapter 2). How we aggregate across annotators determines whose judgments count (Chapter 5). How we design incentives determines what model builders optimize for (Chapter 8). AI measurement science provides tools for making these choices explicit, principled, and auditable — but the choices themselves remain human judgments.

10.2 *Lessons from the Field*

10.2.1 *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 2 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 4) are not incremental. Sang Truong 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.

10.2.2 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 6 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. Sang Truong 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.

10.2.3 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 6 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.

10.3 Open Challenges

10.3.1 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 3; Sang Truong et al. (2025)) is one step in this direction, but a general theory of measurement for mixed response types remains elusive.

10.3.2 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 2) 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?

10.3.3 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 7 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.

10.3.4 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?

10.3.5 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.

10.3.6 Fairness in Evaluation

Evaluation systems can be unfair in multiple ways: items may exhibit differential functioning across model families or deployment contexts (Chapter 6), 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.

10.4 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 2 (IRT models), Chapter 3 (estimation), Chapter 6 (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 4 (CAT, Fisher information), Chapter 2 (IRT), Chapter 7 (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 2 (sufficiency, Rasch), Chapter 5 (item-total correlation), Chapter 6 (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 6 (all validity types, DIF), Chapter 2 (factor models, dimensionality).

Project 6: Scaling Laws through the IRT Lens (⋆⋆)

Replicate and extend the Item Response Scaling Laws framework of Sang Truong 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 2 (IRT), Chapter 3 (Beta-IRT), Chapter 4 (adaptive testing), Chapter 7 (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 9 (adversarial IRT, item banks), Chapter 4 (CAT), Chapter 6 (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 9 (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 9 (PPI, synthetic data), Chapter 5 (judge reliability), Chapter 7 (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 2 (factor models), Chapter 5 (G-theory), Chapter 6 (construct validity), Chapter 4 (design).

10.5 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.

10.5.1 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 6).
- ☐ Distinguish between the construct (latent ability) and the indicator (observed score). A benchmark score is *evidence about* ability, not ability itself (Chapter 2).

10.5.2 Item Development and Calibration

- ☐ Calibrate items using IRT: estimate difficulty, discrimination, and (if applicable) guessing parameters on a diverse set of models (Chapter 3).
- ☐ 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 6).
- ☐ Audit item content for errors: incorrect keys, ambiguous wording, grading bugs, culturally-embedded assumptions.

10.5.3 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 3; Sang Truong 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 4).
- ☐ 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).

10.5.4 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 9).
- ☐ For judge-based evaluation, report inter-rater agreement disaggregated by model and item type. Differential judge error invalidates comparisons (Chapter 9).

10.5.5 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 7).
- ☐ Document known validity threats and limitations. No evaluation is perfectly valid; transparency about limitations is itself a form of quality assurance.

10.5.6 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 6.
- ☐ 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.

10.6 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.

10.7 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 you 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 2
M	Number of items / questions	$\mathbb{N}$	Chapter 2
Y_{ij}	Binary response of model i to item j (0 = incorrect, 1 = correct)	$\{0, 1\}$	Chapter 2
$S_i = \sum_j Y_{ij}$	Sum score (total correct) for model i	$\{0, 1, \dots, M\}$	Chapter 2
$\sigma(x) = \frac{1}{1+e^{-x}}$	Logistic sigmoid function	$(0, 1)$	Chapter 2
$\Phi(x)$	Standard normal CDF	$(0, 1)$	Chapter 2

Item Response Theory

Symbol	Meaning	Domain	Introduced
θ_i or U_i	Latent ability of model i	$\mathbb{R}$	Chapter 2
β_j or V_j	Difficulty of item j	$\mathbb{R}$	Chapter 2
a_j	Discrimination parameter of item j	$\mathbb{R}^+$	Chapter 2
c_j	Guessing (pseudo-chance) parameter of item j	$[0, 1]$	Chapter 2

Symbol	Meaning	Domain	Introduced
d_j	Discrimination parameter (alternative notation, as in 1PL/2PL)	$\mathbb{R}^+$	Chapter 2
z_j	Difficulty parameter (alternative notation)	$\mathbb{R}$	Chapter 2
$I_j(\theta)$	Fisher information for item j at ability θ	$\mathbb{R}^+$	Chapter 4
$\mathcal{I}(\theta)$	Fisher information matrix	$\mathbb{R}^{K\times K}$	Chapter 4

Learning and Estimation

Symbol	Meaning	Domain	Introduced
$\ell(\theta, \beta)$	Log-likelihood function	$\mathbb{R}$	Chapter 3
$\nabla_\theta \ell$	Gradient of log-likelihood w.r.t. ability parameters	$\mathbb{R}^N$	Chapter 3
$\pi(\theta)$	Prior distribution over abilities	–	Chapter 3
$\pi(\beta)$	Prior distribution over difficulties	–	Chapter 3
$\hat{\theta}_{\text{MLE}}$	Maximum likelihood estimate of ability	$\mathbb{R}^N$	Chapter 3
$\hat{\theta}_{\text{MAP}}$	Maximum a posteriori estimate of ability	$\mathbb{R}^N$	Chapter 3
η	Learning rate	$\mathbb{R}^+$	Chapter 3

Reliability

Symbol	Meaning	Domain	Introduced
X_{ij}	Observed score for model i on occasion / item j	$\mathbb{R}$	Chapter 5

Symbol	Meaning	Domain	Introduced
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 6
α_{MH}	Mantel-Haenszel odds ratio	$\mathbb{R}^+$	Chapter 6
λ_k	k-th eigenvalue of the correlation matrix	$\mathbb{R}$	Chapter 6
MNSQ_i	Mean-square fit statistic for item i	$\mathbb{R}^+$	Chapter 6
r_{ij}	Correlation between trait i measured by method j (MTMM)	$[-1, 1]$	Chapter 6

Causality and Distribution Shift

Symbol	Meaning	Domain	Introduced
$\mathrm{do}(X = x)$	Intervention setting variable X to value x	–	Chapter 7
$P^{(s)}, P^{(t)}$	Source (benchmark) / target (deployment) distribution	–	Chapter 7
$\pi_0(a \mid x)$	Logging (benchmark) policy	$[0, 1]$	Chapter 7
$\pi(a \mid x)$	Target (deployment) policy	$[0, 1]$	Chapter 7
$w(x)$	Importance weight $P^{(t)}(x)/P^{(s)}(x)$	$\mathbb{R}^+$	Chapter 7
$\hat{r}(x, a)$	Reward model (e.g., IRT prediction)	$[0, 1]$	Chapter 7
$\hat{V}_{\mathrm{DR}}$	Doubly robust value estimator	$\mathbb{R}$	Chapter 7
$C_\alpha(x)$	Conformal prediction set at level α	subset of $\mathcal{Y}$	Chapter 7

Information and Mechanism Design

Symbol	Meaning	Domain	Introduced
F	Universe of tasks	finite set, $\|F\| = N$	Section 8.2
F_E, F_M	Evaluator's / builder's task sets	$F_E, F_M \subseteq F$	Section 8.2
π_E, π_M	Evaluator's / builder's sampling distributions	distributions on F	Section 8.2
$f(\theta)$	Task performance function	$[0, 1]$	Section 8.2
$u_E(\theta)$	Evaluator's utility: $\sum_{f \in F} f(\theta)$	$\mathbb{R}^+$	Section 8.2
k	Number of sampled evaluation tasks per round	$\mathbb{N}$	Section 8.2
ρ	Distribution correction rate	$(0, 1]$	Section 8.2
γ	Gaming penalty weight	$\mathbb{R}^+$	Section 8.2

Symbol	Meaning	Domain	Introduced
Δ_t	Residual misalignment at round t	$[0, 1]$	Section 8.2
C	Agent cost (principal–agent model)	$\mathbb{R}^+$	Section 8.2
b	Principal's value from agent effort	$\mathbb{R}^+$	Section 8.2

Red-Teaming and Adversarial Evaluation

Symbol	Meaning	Domain	Introduced
$\theta_j^{(\mathrm{adv})}$	Adversarial robustness of model j	$\mathbb{R}$	Chapter 9
$\theta_j^{(\mathrm{std})}$	Standard accuracy ability of model j	$\mathbb{R}$	Chapter 9
$\beta_i^{(\mathrm{atk})}$	Attack strength (difficulty) of adversarial item i	$\mathbb{R}$	Chapter 9
δ	Perturbation applied to an input	$\mathcal{B}_\epsilon$	Chapter 9
$\alpha_{s,\mathcal{D}}$	Attack success probability under criterion s and goal distribution $\mathcal{D}$	$[0, 1]$	Chapter 9
J	Operational judge of attack success	$\{0, 1\}$	Chapter 9
s	Oracle (true) success criterion	$\{0, 1\}$	Chapter 9
K	Number of repeated samples in Top-1 aggregation	$\mathbb{N}$	Chapter 9
$\hat{\mu}_{\mathrm{syn}}$	Mean performance estimated from synthetic items	$[0, 1]$	Chapter 9
$\hat{\mu}_{\mathrm{human}}$	Mean performance estimated from human-authored items	$[0, 1]$	Chapter 9

Symbol	Meaning	Domain	Introduced
$\hat{\mu}_{\text{PPI}}$	Prediction-powered inference estimator	$[0, 1]$	Chapter 9
n	Number of human-evaluated items	$\mathbb{N}$	Chapter 9

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.

Angelopoulos, Anastasios N., Stephen Bates, Emmanuel J. Candès, Michael I. Jordan, and Lihua Lei. 2023. "Prediction-Powered Inference." *Science* 382 (6671): 669–74.

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." In *Social Choice and Welfare*, 48:461–85. 2. Springer.

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." In *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.

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.*

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Blum, Avrim, and Moritz Hardt. 2015. "The Ladder: A Reliable Leaderboard for Machine Learning Competitions." In *Proceedings of the 32nd International Conference on Machine Learning*, 1006–14.

Bommasani, Rishi, Percy Liang, and Tony Lee. 2024. "Holistic Evaluation of Language Models." *Annals of the New York Academy of Sciences*.

Borsboom, Denny. 2005. *Measuring the Mind: Conceptual Issues in Contemporary Psychometrics*. Cambridge, UK: Cambridge University Press.

Bradley, Ralph Allan, and Milton E Terry. 1952. "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." *Biometrika* 39 (3/4): 324–45.

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.

Campbell, Donald T., and Donald W. Fiske. 1959. "Convergent and Discriminant Validation by the Multitrait-Multimethod Matrix." *Psychological Bulletin* 56 (2): 81–105.

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.

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.

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, 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.

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." In *Proceedings of the 28th International Conference on Machine Learning*, 1097–1104.

Dwork, Cynthia, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. 2015. "Generalization in Adaptive Data Analysis and Holdout Reuse." In *Advances in Neural Information Processing Systems*. Vol. 28.

Dwork, Cynthia, and Aaron Roth. 2014. *The Algorithmic Foundations of Differential Privacy*. *Foundations and Trends in Theoretical Computer Science*. Vol. 9. 3–4. Now Publishers.

Embretson, Susan E., and Steven P. Reise. 2000. *Item Response Theory for Psychologists*. Lawrence Erlbaum Associates.

Fleiss, Joseph L. 1971. "Measuring Nominal Scale Agreement Among Many Raters." *Psychological Bulletin* 76 (5): 378–82.

Ganguli, Deep, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, 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.

Goodhart, Charles A. E. 1984. "Problems of Monetary Management: The U.K. Experience." *Monetary Theory and Practice*, 91–121.

Gwet, Kilem Li. 2014. *Handbook of Inter-Rater Reliability*. 4th ed. Advanced Analytics.

Hambleton, Ronald K., and Hariharan Swaminathan. 1985. *Item Response Theory: Principles and Applications*. Boston: Kluwer-Nijhoff.

Hardt, Moritz, Nimrod Megiddo, Christos Papadimitriou, and Mary Wootters. 2016. "Strategic Classification." In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, 111–22. ACM.

Hendrycks, Dan, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. "Measuring Massive Multitask Language Understanding." *Proceedings of the International Conference on Learning Representations (ICLR)*.

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, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, et al. 2023. "C-Eval: A Multi-Level Multi-Discipline Chinese Evaluation Suite for Foundation Models." *Advances in Neural Information Processing Systems*.

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, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Ion Stoica, and Joseph E. Gonzalez. 2024. "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code." *arXiv Preprint arXiv:2403.07974*.

Jimenez, Carlos E., John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. "SWE-Bench: Can Language Models Resolve Real-World GitHub Issues?" *arXiv Preprint arXiv:2310.06770*.

Kamenica, Emir, and Matthew Gentzkow. 2011. "Bayesian Persuasion." *American Economic Review* 101 (6): 2590–2615.

Kane, Michael T. 2006. "Validation." In *Educational Measurement*, edited by Robert L. Brennan, 4th ed., 17–64. Praeger.

Kiefer, Jack. 1959. "Optimum Experimental Designs." *Journal of the Royal Statistical Society: Series B* 21 (2): 272–319.

Kiela, Douwe, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, et al. 2021. "Dynabench: Rethinking Benchmarking in NLP." *arXiv Preprint arXiv:2104.14337*.

Krippendorff, Klaus. 2011. "Computing Krippendorff's Alpha-Reliability." *Departmental Papers (ASC)*.

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, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, et al. 2024. "RewardBench: Evaluating Reward Models for Language Modeling." *arXiv Preprint arXiv:2403.13787*.

Li, Haonan, Yixuan Zhang, Fajri Koto, Yejin Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. 2023. "CMMLU: Measuring Massive Multitask Language Understanding in Chinese." *arXiv Preprint arXiv:2306.09212*.

Linden, Wim J. van der. 2006. "Optimal Test Design." *Handbook of Statistics* 26: 575–95.

Lord, Frederic M., and Melvin R. Novick. 1968. *Statistical Theories of Mental Test Scores*. Reading, MA: Addison-Wesley.

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*.

Manheim, David, and Scott Garrabrant. 2018. "Categorizing Variants of Goodhart's Law." *arXiv Preprint arXiv:1803.04585*.

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.

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.

METR. 2025. "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity." *arXiv Preprint arXiv:2507.09089*. https://arxiv.org/abs/2507.09089.

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.

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, Karina Nguyen, Edwin Chen, Scott Heiner, Craig Pettit, et al. 2022. "Red Teaming Language Models with Language Models." *arXiv Preprint arXiv:2202.03286*.

Peters, Jonas, Dominik Janzing, and Bernhard Schölkopf. 2017. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press.

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.

Ribeiro, Marco Tulio, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList." In *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.

Salaudeen, Oluwadara, Anka Reuel, Amina Ahmed, Tanmay Bedi, Cassidy Robertson, Aishwarya Sundar, Ben Domingue, Serena Wang, and Sanmi Koyejo. 2025. "Measurement to Meaning: A Validity-Centered Framework for AI Evaluation." *arXiv Preprint arXiv:2505.10573*. https://arxiv.org/abs/2505.10573.

Schaeffer, Rylan, Joshua Kazdan, John Hughes, Jordan Juravsky, Sara Price, Aengus Lynch, Erik Jones, Robert Kirk, Azalia Mirhoseini, and Sanmi Koyejo. 2025. "How Do Large Language Monkeys Get Their Power (Laws)?" *arXiv Preprint arXiv:2502.17578*.

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.

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.

Sugiyama, Masashi, Taiji Suzuki, and Takafumi Kanamori. 2012. *Density Ratio Estimation in Machine Learning*. Cambridge University Press.

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.

Truong, Sang T., Yuheng Tu, Michael Hardy, Anka Reuel, Zeyu Tang, Jirayu Burapacheep, Jonathan Perera, 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, Son, Serena Wang, Hoda Heidari, and Rishi Bommasani. 2025. "Incentive-Aligned Evaluation via Private Benchmark." *arXiv Preprint arXiv:2506.00000*.

Vapnik, Vladimir N. 1998. *Statistical Learning Theory*. New York: Wiley.

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, Angelina Wang, Chad Atalla, Solon Barocas, Su Lin Blodgett, et al. 2025. "Position: Evaluating Generative AI Systems Is a Social Science Measurement Challenge." *arXiv Preprint arXiv:2502.00561*.

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*.

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*. Chicago: MESA Press.

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.

Zellers, Rowan, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. "HellaSwag: Can a Machine Really Finish Your Sentence?" In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4791–4800.

Zhang, Shirley, and Moritz Hardt. 2024. "The Inherent Tradeoffs in LLM Benchmarks." *arXiv Preprint*.

Zheng, Lianmin, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, et al. 2023. "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena." *Advances in Neural Information Processing Systems*.

Zumbo, Bruno D. 1999. *A Handbook on the Theory and Methods of Differential Item Functioning (DIF)*. Ottawa: National Defense Headquarters.

Index