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

Introduction

Progress in artificial intelligence depends on knowing what our systems can do, how well they can do it, and under what conditions their behavior changes. Evaluation is therefore not an afterthought to AI development but its epistemic foundation. Every claim of progress—whether about improved reasoning, better alignment, or broader capability—rests on some act of measurement. Yet despite the centrality of evaluation, the field's tools for measurement remain strikingly underdeveloped. Benchmarks proliferate faster than we can understand them, and leaderboards offer scores without scales, turning scientific assessment into a race of numbers detached from theory.

Contemporary evaluation practice in AI largely relies on finite collections of datasets and metrics—benchmarks that serve as de facto instruments of measurement. These instruments are often designed without formal notions of validity, reliability, or calibration. A model's average accuracy across a dataset says little about why it succeeds, where it fails, or how its abilities generalize beyond the test. The result is an evaluation ecosystem that produces motion without understanding. Without a coherent measurement framework, we risk mistaking leaderboard ascent for scientific progress.

Other sciences have faced similar crises of interpretation and responded by formalizing the theory of measurement. Psychology turned to psychometrics, developing Item Response Theory (IRT) and latent variable modeling to distinguish true ability from test difficulty. Education systems built statistical foundations for comparing learners across tests, time, and populations. The physical sciences standardized their instruments and units to make measurement traceable and comparable across laboratories. Each of these fields transformed ad hoc evaluation into measurement science—a discipline grounded in inference, uncertainty, and calibration. AI now stands at a similar inflection point.

AI Measurement Sciences (AIMS) seeks to provide this missing foundation. It treats AI evaluation as an inferential problem: given observed responses of models to tasks, what can we infer about their latent capabilities, the properties of the tasks, and the conditions of generalization? It asks how to design evaluation systems that are reliable (stable under sampling and perturbation), valid (measuring intended constructs rather than artifacts), and interpretable (enabling meaningful comparison across time, domains, and model families). The central research questions are therefore not only empirical but epistemological:

- How can we represent and estimate the latent constructs underlying AI performance?
- How can we quantify uncertainty, bias, and contamination in existing evaluation systems?

– What are the statistical and organizational conditions under which measurement becomes trustworthy enough to guide AI development and governance?

This dissertation argues that answering these questions requires a probabilistic science of evaluation—one that unites methods from psychometrics, statistics, and machine learning. The goal is not merely to build better benchmarks but to establish a framework for scientific decision making about AI systems: when to trust, when to doubt, and how to act.

1 Foundations

2 Foundations of Measurement

Intended Learning Outcomes

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

1. **Articulate** Borsboom's realist framework for validity and explain why measurement requires warrant inference about latent constructs.
2. **Distinguish** between Item Response Theory, factor models, paired comparison systems (Elo, Bradley-Terry), and network models (GGM, Ising).
3. **Explain** why the Rasch model holds a special status as "the measurement model" through the sufficiency of sum scores, specific objectivity, and test-free measurement.
4. **Derive** the sufficiency theorem for the Rasch model and explain its implications for AI benchmark evaluation.
5. **Compare** the prescriptive (Rasch school) and descriptive (general IRT) approaches to measurement and articulate when each is appropriate.
6. **Trace** the historical development from Thurstone (1927) through Rasch (1960) to modern network psychometrics.
7. **Connect** classical measurement concepts (reliability, validity, dimensionality) to contemporary AI benchmark evaluation.
8. **Apply** measurement theory to analyze whether AI benchmarks satisfy the requirements for scientific measurement.
9. **Implement** basic IRT models in Python and visualize item characteristic curves.
10. **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

Throughout this chapter, we use the following conventions:

Symbol	Meaning	Domain
θ_i or U_i	Latent ability of person/model i	$\mathbb{R}$
β_j or V_j	Difficulty of item j	$\mathbb{R}$

a_j	Discrimination parameter of item j	$\mathbb{R}^+$
c_j	Guessing parameter of item j	$[0, 1]$
Y_{ij}	Binary response (0 = incorrect, 1 = correct)	$\{0, 1\}$
$S_i = \sum_j Y_{ij}$	Sum score (total correct) for person i	$\{0, 1, \ldots, M\}$
N	Number of persons/models	$\mathbb{N}$
M	Number of items/questions	$\mathbb{N}$
$\sigma(x) = \frac{1}{1+e^{-x}}$	Logistic sigmoid function	$(0, 1)$
$\Phi(x)$	Standard normal CDF	$(0, 1)$

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 $i \in \{1, \ldots, N\}$ represents a *test taker* (in AI: a model)
- Each column $j \in \{1, \ldots, M\}$ 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.

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

```
})
```

```
#| label: response-matrix-visualization
#| autorun: true
#| fig-cap: "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

# 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 Borsboom's Warrant Inference Framework

Before we can measure something, we must understand what measurement *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.

2.2.1 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?).

2.2.2 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.

2.2.3 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.

2.2.4 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 → better performance on logic puzzles
- Higher reasoning ability → better performance on mathematical proofs
- Higher reasoning ability → correlation with general intelligence
- Higher reasoning ability → 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.

2.3 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.

2.3.1 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.

2.3.1.1 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.

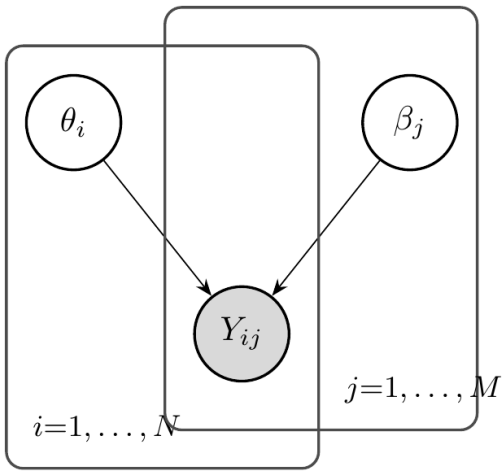

Figure 2.1
Plate diagram for the Rasch model. Shaded nodes are observed; open nodes are latent. Plates indicate replication over persons (i) and items (j).

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.

```
#| 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()
```

2.3.1.2 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.

```
#| 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]

```

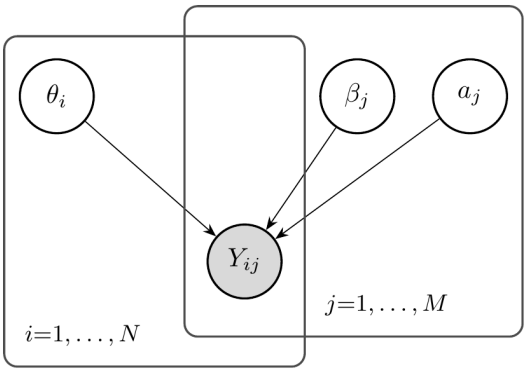

Figure 2.2
Plate diagram for the 2PL model. Each item now has both a difficulty β_j and a discrimination a_j parameter.

```
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()
```

2.3.1.3 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.

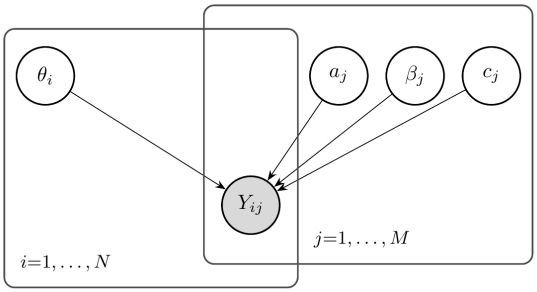

Figure 2.3
Plate diagram for the 3PL model. The guessing parameter c_j sets a lower asymptote on the response probability.

```
#| 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

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()
```

2.3.2 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.

2.3.2.1 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.

2.3.2.2 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.

2.3.2.3 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 $Z_j = -\beta_j$.

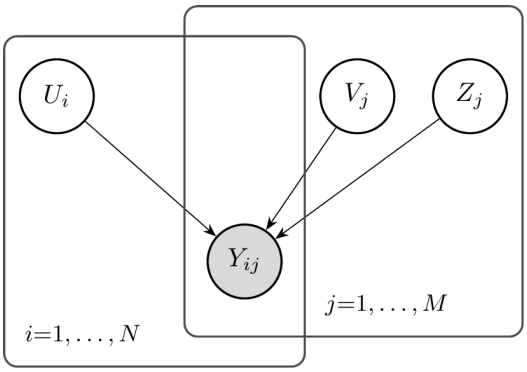

Figure 2.4
Plate diagram for the logistic factor model. The latent factor vector U_i interacts with item-specific loadings V_j and intercepts Z_j to produce responses.

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

2.3.2.4 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:

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

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

The 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.

2.3.2.5 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.

2.3.3 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.

2.3.3.1 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.

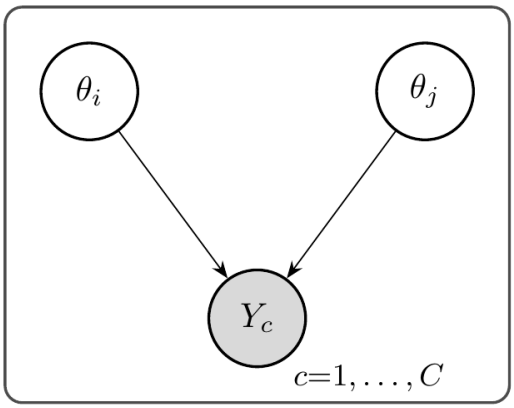

Figure 2.5
Plate diagram for the Bradley-Terry model. A comparison outcome Y_c depends on the strengths θ_i and θ_j of the two competitors in each pair.

2.3.3.2 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.

2.3.3.3 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.

2.3.4 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.

2.3.4.1 The Gaussian Graphical Model (GGM)

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

DEFINITION: GAUSSIAN GRAPHICAL MODEL

Variables $X = (X_1, \ldots, X_M)$ 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.

2.3.4.2 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.

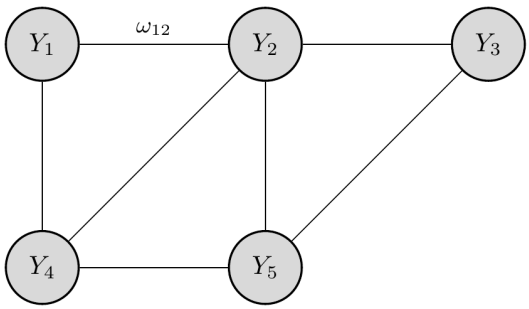

Figure 2.6
Graphical model for the Ising model. Unlike latent variable models, all nodes are observed and correlations arise from direct pairwise connections ω_{jk}.

2.3.4.3 *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.

2.3.5 *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:

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

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.

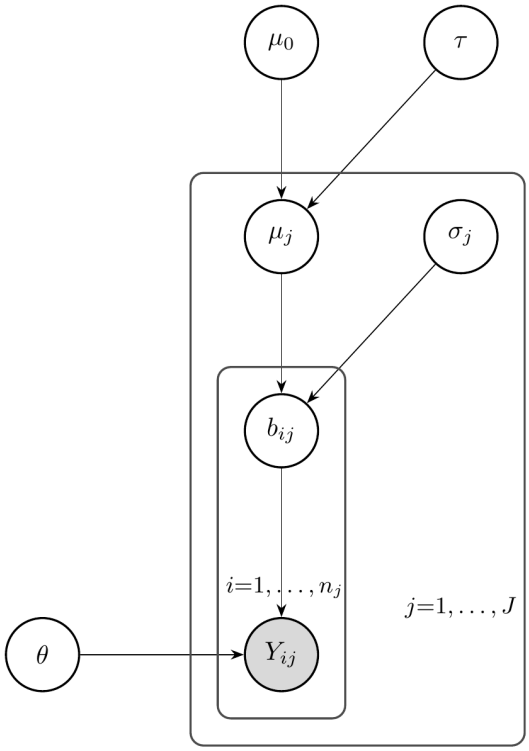

Figure 2.7
Plate diagram for the hierarchical IRT model. Item difficulties b_{ij} are drawn from benchmark-level distributions parameterized by μ_j and σ_j, which may themselves be drawn from domain-level hyperparameters μ_0 and τ. Nested plates reflect the hierarchical data structure.

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 exchangeable 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.

2.4 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.

2.4.1 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 $Y_i = (Y_{i1}, \ldots, Y_{iM})$ 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.

2.4.1.1 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.

2.4.2 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.

2.4.3 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.

2.4.3.1 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.

2.4.4 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.

2.4.4.1 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.

2.4.4.2 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.

2.4.4.3 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

Question	Rasch School Answer	General IRT Answer
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.

2.5 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.

2.5.1 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.

2.5.2 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.

2.5.3 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.

2.5.4 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.

2.5.5 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.

2.6 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.

2.6.1 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	–
Adaptive testing (CAT)	Efficient evaluation	–

2.6.2 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?

2.6.3 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.

2.6.4 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.

2.7 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 is about truth, not evidence**. Following Borsboom, validity means that the attribute exists and causally produces variation in scores. Evidence supports validity claims but does not constitute validity.
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, and computerized adaptive testing. Also introduces generalization experiments with various masking schemes.
- **Chapter 3 (Design)**: Applies measurement principles to benchmark design, addressing how to construct valid and reliable AI evaluations.
- **Chapter 4 (Conclusion)**: Synthesizes the measurement science approach and discusses future directions for AI evaluation.

The measurement concepts from this chapter recur throughout. When we ask whether a benchmark is "valid," we mean validity in Borsboom's sense. 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.

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.

```
# 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.

```
# Given: Comparison data as list of (winner, loser) pairs
# Task: Estimate strength parameters via maximum likelihood
#
# The likelihood for comparison (i beats j) is:
# P(i > j) = exp(theta_i) / (exp(theta_i) + exp(theta_j))
#          = sigmoid(theta_i - theta_j)

import numpy as np
from scipy.optimize import minimize

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

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

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

```
# 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: Borsboom argues that validity is about truth, not evidence. How does this change how we should think about AI benchmark validity? Can a benchmark be "valid enough" for practical purposes even if we cannot prove the underlying construct exists?

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?

2.9 Bibliographic Notes

2.9.1 Validity and Measurement Philosophy

The realist framework for validity originates with Borsboom's influential paper "The Concept of Validity" (Borsboom, Mellenbergh, & van Heerden, 2004) and his book *Measuring the Mind* (2005). For a comprehensive treatment, see *Frontiers of Test Validity Theory* (Markus & Borsboom, 2013). The classic reference on validity as evidence accumulation is Messick's chapter in *Educational Measurement* (1989).

2.9.2 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.

2.9.3 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.

2.9.4 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).

2.9.5 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.

2.9.6 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).

3 Learning

4 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. **Design** a Computerized Adaptive Testing (CAT) procedure using Fisher information for item selection.
10. **Apply** MLE, Bayesian, and CAT methods to real AI benchmark data and compare their efficiency.

Suggested Lecture Plan

This chapter can be covered in **3–4 lectures** (75–90 minutes each):

Lecture 1: Foundations of Estimation

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

Lecture 2: Advanced Estimation Methods

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

Lecture 3: Bayesian Approaches

- Prior specification for IRT (15 min)
- MAP estimation (20 min)
- MCMC for IRT (30 min)
- Regularization as Bayesian prior (10 min)

Lecture 4: Active Learning

- CAT framework and Fisher information (25 min)
- D-optimality and item selection (20 min)
- Stopping rules and efficiency (15 min)
- Hands-on: CAT simulation (15 min)

Notation

Building on Chapter 1, we use the following additional notation:

Symbol	Meaning	Domain
$\ell(\theta, \beta)$	Log-likelihood function	$\mathbb{R}$
$\nabla_\theta \ell$	Gradient w.r.t. ability parameters	$\mathbb{R}^N$
$\mathcal{I}(\theta)$	Fisher information matrix	$\mathbb{R}^{N \times N}$
$I_j(\theta)$	Fisher information for item j	$\mathbb{R}^+$
$\pi(\theta)$	Prior distribution over abilities	-
$\pi(\beta)$	Prior distribution over difficulties	-
$\hat{\theta}_{\text{MLE}}$	Maximum likelihood estimate	$\mathbb{R}^N$
$\hat{\theta}_{\text{MAP}}$	Maximum a posteriori estimate	$\mathbb{R}^N$
η	Learning rate	$\mathbb{R}^+$

Video Overview

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

../animations/ch2/chapter2_narrated.mp4

4.1 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.
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 covers two complementary paradigms for learning these parameters:

- **Passive learning**: Given a fixed dataset, estimate all parameters simultaneously. This includes maximum likelihood estimation (MLE), expectation-maximization (EM), and Bayesian inference.
- **Active learning**: Sequentially select which questions to administer based on current estimates, updating parameters after each response. Computerized Adaptive Testing (CAT) is the primary example.

4.2 Maximum Likelihood Estimation

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

4.2.1 The Likelihood Function

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

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

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:

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

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

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

This is the objective function we want to maximize.

4.2.2 Gradient Derivation

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

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

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

Intuitive Interpretation of the Gradient

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

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

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

4.2.3 Implementation with Gradient Descent

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

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

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

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

import numpy as np
import matplotlib.pyplot as plt

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

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

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

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

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

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

```
33
34 # Raw response matrix
35 axes[0].imshow(Y, aspect='auto', cmap='Blues')
36 axes[0].set_xlabel('Questions')
37 axes[0].set_ylabel('Models')
38 axes[0].set_title('Raw Response Matrix')
39
40 # Sorted by ability and difficulty
41 row_order = np.argsort(Y.mean(axis=1))[::-1]
42 col_order = np.argsort(Y.mean(axis=0))[::-1]
43 Y_sorted = Y[row_order][:, col_order]
44 axes[1].imshow(Y_sorted, aspect='auto', cmap='Blues')
45 axes[1].set_xlabel('Questions (sorted by difficulty)')
46 axes[1].set_ylabel('Models (sorted by ability)')
47 axes[1].set_title('Sorted Response Matrix')
48
49 plt.tight_layout()
50 plt.show()
```

Now we implement MLE via gradient descent:

```
1  #| label: mle-gradient-descent
2  #| autorun: true
3  #| fig-cap: "Convergence of gradient descent for Rasch model MLE."
4
5  def rasch_log_likelihood(theta, beta, Y):
6      """Compute Rasch model log-likelihood."""
7      logits = theta[:, None] - beta[None, :]
8      ll = (Y * logits - np.log(1 + np.exp(np.clip(logits, -500, 500)))).sum()
9      return ll
10
11 def rasch_gradients(theta, beta, Y):
12     """Compute gradients for theta and beta."""
13     P = sigmoid(theta[:, None] - beta[None, :])
14     grad_theta = (Y - P).sum(axis=1)
15     grad_beta = (P - Y).sum(axis=0)
16     return grad_theta, grad_beta
17
18 # Initialize parameters at zero
19 theta_hat = np.zeros(N)
20 beta_hat = np.zeros(M)
21
22 # Gradient ascent
23 learning_rate = 0.01
24 n_iterations = 500
25 ll_history = []
26
27 for iteration in range(n_iterations):
```

```
    # Compute gradients
    grad_theta, grad_beta = rasch_gradients(theta_hat, beta_hat, Y)

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

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

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

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

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

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

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

plt.tight_layout()
plt.show()

# Correlation with true values
```

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

4.2.4 *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.

4.2.5 *L-BFGS Optimization*

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

```
#| 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}")
```

4.3 Joint, Conditional, and Marginal MLE

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

4.3.1 Joint MLE (JMLE)

JMLE simultaneously estimates all parameters by maximizing Equation 4.3. While intuitive, JMLE suffers from the *incidental parameter problem*: 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.

4.3.2 Conditional MLE (CMLE)

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

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

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

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

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

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

4.3.3 Marginal MLE (MMLE)

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

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

The marginal likelihood integrates out the person parameters:

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

This approach:

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

Comparison of MLE Approaches

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

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

4.4 The EM Algorithm

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

4.4.1 The EM Framework

The EM algorithm iterates between two steps:

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

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

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

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

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

4.4.2 EM for the Rasch Model

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

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

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

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

M-step: Update each item parameter by solving:

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

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

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

from numpy.polynomial.hermite import hermgauss

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

    Parameters
```

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

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

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

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

    ll_history = []

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

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

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

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

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

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

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

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

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

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

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

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

```
    theta_hat = (posterior * nodes).sum(axis=1)

    return theta_hat, beta, ll_history

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

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

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

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

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

plt.tight_layout()
plt.show()

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

4.4.3 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.

4.4.3.1 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 $[0, 1]$

4.4.3.2 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.

4.5 Bayesian Inference

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

4.5.1 Prior Specification

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

STANDARD PRIORS FOR IRT

For abilities (persons/models):

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

For difficulties (items/questions):

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

For discrimination (2PL model):

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

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.

4.5.2 Posterior Computation

Bayes' theorem gives us the posterior distribution:

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

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

4.5.3 MAP Estimation

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

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

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

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

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

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

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

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

```
    return -(ll + log_prior_theta + log_prior_beta)

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

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

    return np.concatenate([grad_theta, grad_beta])

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

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

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

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

# Abilities
axes[0].scatter(theta_true_centered, theta_lbfgs, alpha=0.5, label='MLE', s=30)
axes[0].scatter(theta_true_centered, theta_map, alpha=0.5, label='MAP', s=30)
axes[0].plot([-3, 3], [-3, 3], 'k--', alpha=0.5)
axes[0].set_xlabel('True ability')
axes[0].set_ylabel('Estimated ability')
axes[0].set_title('Ability Estimates: MLE vs MAP')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Difficulties
axes[1].scatter(beta_true_centered, beta_lbfgs, alpha=0.5, label='MLE', s=30)
axes[1].scatter(beta_true_centered, beta_map, alpha=0.5, label='MAP', s=30)
axes[1].plot([-4, 4], [-4, 4], 'k--', alpha=0.5)
axes[1].set_xlabel('True difficulty')
```

```
axes[1].set_ylabel('Estimated difficulty')
axes[1].set_title('Difficulty Estimates: MLE vs MAP')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

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

BAYESIAN SHRINKAGE

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

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

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

4.5.4 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

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

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

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

        # Accept or reject
        if np.log(np.random.random()) < log_alpha:
            theta = theta_prop
            beta = beta_prop
            current_lp = prop_lp
            if s >= n_warmup:
                n_accept += 1

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

    acceptance_rate = n_accept / n_samples
```

```
    if verbose:
        print(f"Acceptance rate: {acceptance_rate:.3f}")

    return theta_samples, beta_samples, acceptance_rate

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

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

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

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

plt.tight_layout()
plt.show()

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

```

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

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

4.6 Regularization and Model Selection

4.6.1 L2 Regularization as Bayesian Prior

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

The regularized objective is:

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

Regularization prevents overfitting, especially when:

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

4.6.2 Cross-Validation for Hyperparameter Selection

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

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

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

    # Convert lambda to prior std
```

```
    if sigma_theta is None:
        sigma_theta = 1 / np.sqrt(lambda_param + 1e-10)
    if sigma_beta is None:
        sigma_beta = 1 / np.sqrt(lambda_param + 1e-10)

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

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

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

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

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

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

    return ll_test / n_test  # Average log-likelihood

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

    # 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}")
```

4.7 Active Learning: Computerized Adaptive Testing

So far we have discussed *passive learning*: given a fixed dataset, estimate all parameters. But in many AI evaluation scenarios, we can choose which questions to ask. This is *active learning*, and Computerized Adaptive Testing (CAT) is its primary instantiation in psychometrics.

4.7.1 The CAT Framework

The key insight of CAT is that not all questions are equally informative for all test-takers. A very easy question provides little information about a high-ability model—we already know it will likely answer correctly. Similarly, a very hard question provides little information about a low-ability model.

The most informative questions are those where the model has roughly a 50% chance of success. CAT iteratively:

1. **Select** the most informative question given current ability estimate
2. **Administer** the question and observe the response

3. **Update** the ability estimate based on the response
4. **Check** if stopping criterion is met; if not, return to step 1

> **Why Fisher Information?**
>
> Fisher information measures how much a response to item j tells us about θ:
>
> – **High information**: The item is well-matched to the ability level
> – **Low information**: The item is too easy or too hard
>
> Intuitively, asking a genius to solve $1 + 1$ or a beginner to prove the Riemann hypothesis provides little information. The most informative items are those where the model has about 50% chance of success.

4.7.2 Fisher Information for Item Selection

The Fisher information for item j at ability θ in the Rasch model is:

$$I_j(\theta) = P_j(\theta) \cdot (1 - P_j(\theta)) \tag{4.10}$$

where $P_j(\theta) = \sigma(\theta - \beta_j)$.

This is maximized when $P_j(\theta) = 0.5$, which occurs when $\theta = \beta_j$. Thus, the optimal item to administer is the one whose difficulty most closely matches the current ability estimate.

```
#| label: fisher-information
#| autorun: true
#| fig-cap: "Fisher information as a function of ability for items of different
 ↪  difficulties."

# Plot Fisher information curves
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'$\\beta_j = {beta_j}$')

axes[0].set_xlabel('Ability ($\\theta$)')
axes[0].set_ylabel('Fisher Information')
```

```
axes[0].set_title('Item Information Curves')
axes[0].legend()
axes[0].grid(True, alpha=0.3)
axes[0].axvline(0, color='gray', linestyle=':', alpha=0.5)

# Cumulative information from multiple items
theta_test = 0.5  # Example ability
n_items = 20

# Adaptive selection: choose items closest to current estimate
beta_available = beta_true.copy()
adaptive_info = [0]
theta_estimate = 0  # Start with prior mean

for t in range(n_items):
    # Select item with difficulty closest to current estimate
    distances = np.abs(beta_available - theta_estimate)
    best_idx = np.argmin(distances)
    beta_selected = beta_available[best_idx]

    # Information from this item
    P = sigmoid(theta_test - beta_selected)
    info = P * (1 - P)
    adaptive_info.append(adaptive_info[-1] + info)

    # Remove selected item
    beta_available = np.delete(beta_available, best_idx)

    # Update estimate (simplified: use true ability for demo)
    theta_estimate = theta_test  # In practice, we'd use MAP update

# Random selection
np.random.seed(42)
random_order = np.random.permutation(len(beta_true))[:n_items]
random_info = [0]
for j in random_order:
    P = sigmoid(theta_test - beta_true[j])
    info = P * (1 - P)
    random_info.append(random_info[-1] + info)

axes[1].plot(range(n_items + 1), adaptive_info, 'g-', linewidth=2, label='Adaptive')
axes[1].plot(range(n_items + 1), random_info, 'b-', linewidth=2, label='Random')
axes[1].set_xlabel('Number of Items')
axes[1].set_ylabel('Cumulative Fisher Information')
axes[1].set_title(f'Information Accumulation ($\\theta = {theta_test}$)')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
```

```
plt.show()
```

4.7.3 CAT Implementation

Let us implement a complete CAT procedure:

```
#| label: cat-simulation
#| autorun: true
#| fig-cap: "CAT efficiency compared to random item selection for reaching target
 ↪  reliability."

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 results
    """
    M = len(beta)

    # Track administered items and responses
    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)
        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 using MAP (Newton-Raphson)
        for _ in range(10):
            P_vec = sigmoid(theta_hat - np.array([beta[j] for j in administered]))

            # Gradient: sum of residuals minus prior contribution
            grad = np.sum(np.array(responses) - P_vec) - theta_hat / prior_var

            # Hessian: negative sum of P(1-P) minus prior contribution
            hess = -np.sum(P_vec * (1 - P_vec)) - 1 / prior_var

            if abs(hess) > 1e-10:
                theta_hat = theta_hat - grad / hess

        # Compute posterior variance (inverse of observed information + prior
    ↪ precision)
        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)
        se = np.sqrt(posterior_var)

        # Reliability: proportion of variance explained
        # R = 1 - error_var / total_var, where total_var = 1 (prior)
        reliability = 1 - posterior_var / prior_var

        theta_history.append(theta_hat)
        reliability_history.append(reliability)
        se_history.append(se)

        # Check stopping criterion
```

```
        if reliability >= reliability_threshold:
            break

    return {
        'theta_hat': theta_hat,
        'theta_true': theta_true_i,
        'n_items': len(administered),
        'administered': administered,
        'responses': responses,
        'reliability_history': reliability_history,
        'theta_history': theta_history,
        'se_history': se_history,
        'final_reliability': reliability_history[-1],
        'final_se': se_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)

    # Random order
    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:
        # Simulate response
        P_true = sigmoid(theta_true_i - beta[j])
        response = int(np.random.random() < P_true)

        administered.append(j)
        responses.append(response)

        # Update ability estimate
        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
```

```
        # Posterior variance and reliability
        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 for multiple test-takers
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: average items needed
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')

# Add values on bars
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 trajectories for a single example
example_idx = 50
example_cat = cat_results[example_idx]
example_random = random_results[example_idx]

axes[1].plot(example_random['reliability_history'], 'b-', linewidth=2, label='Random')
axes[1].plot(example_cat['reliability_history'], 'g-', linewidth=2, label='CAT')
axes[1].axhline(0.95, color='r', linestyle='--', linewidth=1.5, label='Threshold
 ↪  (0.95)')
axes[1].set_xlabel('Number of items administered')
axes[1].set_ylabel('Reliability')
axes[1].set_title(f'Reliability Growth (example: $\\theta$ =
 ↪  {theta_test_sample[example_idx]:.2f})')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

# Histogram of items needed
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()

# Summary statistics
efficiency_gain = (np.mean(random_items) - np.mean(cat_items)) / np.mean(random_items)
 ↪  * 100

print(f"\nSummary:")
print(f"  Random selection: {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")
```

4.7.4 Stopping Rules

CAT can use various stopping criteria:

1. **Reliability threshold**: Stop when measurement precision reaches a target (e.g., $R \geq 0.95$)
2. **Standard error threshold**: Stop when $SE(\hat{\theta}) \leq 0.3$
3. **Fixed length**: Administer exactly K items
4. **Information threshold**: Stop when additional items would provide negligible information

For AI evaluation, practical constraints also matter:

- **Cost**: Each API call has a cost
- **Time**: Evaluation must complete within a deadline
- **Contamination**: Administering too many items risks benchmark leakage

> **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 be deterministic depending on temperature and sampling settings.
>
> For deterministic evaluation (temperature=0), CAT works 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.

4.8 Generalization Experiments

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.

4.8.1 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

Masking Type	Train Set	Test Set	Purpose
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.

4.8.2 Implementation of Masking Functions

```
import torch

def random_mask(data_idtor, pct=0.8):
    """Entry-wise random masking."""
    train_idtor = torch.bernoulli(data_idtor * pct).int()
    test_idtor = data_idtor.int() - train_idtor
    return train_idtor, test_idtor

def model_mask(data_idtor, pct_models=0.8, exposure_rate=0.3):
    """Row holdout: hold out unseen models."""
    train_row_mask = torch.bernoulli(torch.ones(data_idtor.shape[0]) *
    ↪ pct_models).bool()
    train_idtor = torch.zeros_like(data_idtor).int()
    train_idtor[train_row_mask, :] = data_idtor[train_row_mask, :]
    train_idtor[~train_row_mask, :], _ = random_mask(data_idtor[~train_row_mask, :],
    ↪ pct=exposure_rate)
    test_idtor = data_idtor - train_idtor
    return train_idtor, test_idtor

def item_mask(data_idtor, pct_items=0.8, exposure_rate=0.3):
    """Column holdout: hold out unseen items."""
    train_col_mask = torch.bernoulli(torch.ones(data_idtor.shape[1]) *
    ↪ pct_items).bool()
    train_idtor = torch.zeros_like(data_idtor).int()
    train_idtor[:, train_col_mask] = data_idtor[:, train_col_mask]
    train_idtor[:, ~train_col_mask], _ = random_mask(data_idtor[:, ~train_col_mask],
    ↪ pct=exposure_rate)
    test_idtor = data_idtor - train_idtor
    return train_idtor, test_idtor

def L_mask(data_idtor, pct_models=0.8, pct_items=0.8):
    """Row-column block (L-mask): compositional generalization."""
    train_row_mask = torch.bernoulli(torch.ones(data_idtor.shape[0]) *
    ↪ pct_models).bool()
```

```
    train_col_mask = torch.bernoulli(torch.ones(data_idtor.shape[1]) *
↪ pct_items).bool()
    train_idtor = torch.zeros_like(data_idtor).int()
    train_idtor[train_row_mask][:, train_col_mask] = data_idtor[train_row_mask][:,
↪ train_col_mask]
    test_idtor = data_idtor - train_idtor
    test_idtor[train_row_mask, :] = 0
    test_idtor[:, train_col_mask] = 0
    return train_idtor, test_idtor
```

4.8.3 Two-Stage Training for Holdout Generalization

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

4.8.3.1 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)
```

4.8.3.2 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.

4.8.4 Evaluation Across Masking Schemes

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

```
from torchmetrics import AUROC

masking_schemes = {
    "entry_random": random_mask,
    "row_holdout": model_mask,
    "col_holdout": item_mask,
    "L_mask": L_mask,
}

results = {}
auroc = AUROC(task="binary")

for name, mask_fn in masking_schemes.items():
    train_mask, test_mask = mask_fn(data_idtor)

    # Train model (with two-stage for row/col holdout)
    model = train_with_appropriate_stages(Y, train_mask, test_mask, name)

    # Evaluate on held-out entries
```

```
    P_hat = model().detach()
    auc = auroc(P_hat[test_mask.bool()], Y[test_mask.bool()])
    results[name] = auc.item()
    print(f"{name}: AUC = {auc:.3f}")
```

The factor model typically achieves AUC of 92–97% on random masking across benchmarks, demonstrating strong predictive power. Performance on row and column holdout tests the model's ability to generalize to new models and new items, respectively.

4.9 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. **Adaptive Testing for AI**: Current AI benchmarks test all models on all questions. What are the practical challenges in implementing CAT for AI evaluation? Consider: determinism of model responses, cost of API calls, benchmark contamination.
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?

4.10 Bibliographic Notes

4.10.1 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 (?).

4.10.2 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.

4.10.3 EM Algorithm

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

4.10.4 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 (?).

4.10.5 Computerized Adaptive Testing

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

4.10.6 Optimization Methods

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

4.11 Exercises

4.11.1 Theoretical Exercises

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

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

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

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

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

4.11.2 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 ($\star\star\star$): Extend the CAT simulation to handle a multidimensional factor model with $K = 2$ dimensions. Implement D-optimal item selection using $j^* = \arg\max_j \det(\sum_\tau I_j^{(\tau)})$.

4.11.3 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: Design a stopping rule for CAT that balances measurement precision with evaluation cost. How would you adapt this for AI evaluation where API calls have monetary costs?

Exercise 2.11: 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 and quantify the effect.

5 Design

6 GENERALIZATION

Large language models (LLMs) are often evaluated by running them on benchmarks and asking an **AI judge** to score their answers.
However, judging introduces bias and high cost — each (model, question) pair must be queried and scored.

This tutorial walks through an alternative framework — **Prediction-Powered Evaluation (PPE)** — which predicts correctness *without running models or judges*.
It combines **factor analysis** and **semantic prediction** to estimate correctness probabilities for unseen questions or unseen models.

6.1 Motivation

6.1.1 Limitations of Judge-Based Evaluation

Judge-based approaches are expensive and biased by surface-level *style features* (e.g., bulleting, verbosity).
We formalize two approaches to measuring correctness:

$$p_\theta(Y_{ij} = 1 \mid i, j, D_{\text{train}}) = \sigma(H_{ij}(\theta))$$

and the *judging* variant:

$$p_\theta(Y_{ij} = 1 \mid i, j, k, D_{\text{train}}) = \mathbb{E}_k[p_\theta(Y_{ij} = 1 \mid i, j, k, D_{\text{train}})]$$

Let S denote style (e.g., response length).
Then the judge model induces a **bias pathway** $S \to R \to \hat{Y}_{\text{judge}}$,
while the prediction-powered correctness model $\hat{Y}_{\text{corr}}$ remains unbiased:

$$\text{Bias}_{\text{judge}}(s) = E[\hat{Y}_{\text{judge}} - Y^* \mid S = s], \quad \text{Bias}_{\text{corr}}(s) = 0.$$

This framework enables **cost-efficient, style-invariant evaluation**, avoiding the stylistic confounds of AI judges.

6.1.2 The Hardness of Mapping from Semantics to Behavior

Even if we could perfectly represent question meaning, **semantic similarity does not guarantee behavioral similarity**.
Two questions that appear linguistically close may elicit very different correctness patterns across models.

We compare:

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

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

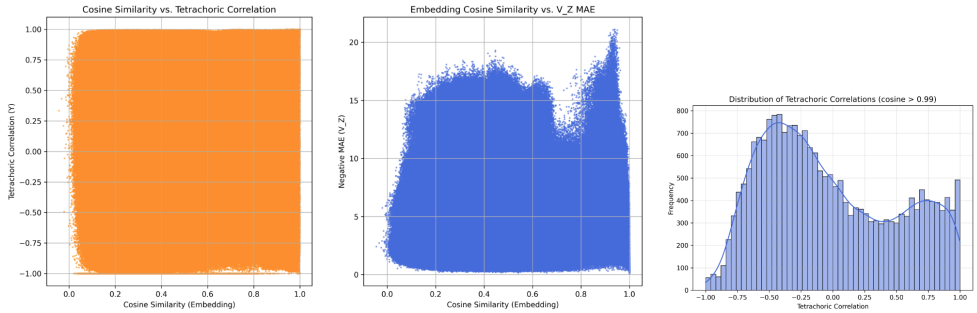

Figure 6.1
Semantic vs behavioral similarity between question pairs. Even near-identical embeddings (cosine > 0.99) show random behavioral correlations.

As shown in Figure 4.1,
there is **no consistent relationship** between these two measures — even when $\cos(E_i, E_j) > 0.99$, the behavioral correlation can range from −1 to +1.
This randomness reveals that **semantic embeddings are poor instruments** for explaining or predicting response behavior.

Observation: Semantically similar questions (cosine > 0.99) exhibit nearly random behavioral correlations (−1 to +1),
showing that linguistic proximity does **not** imply behavioral equivalence.

6.2 Stage 1 — Factor Model Pretraining

We first learn latent behavioral factors (U, V, Z) from response data Y_{ij}.

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

Each model i has latent ability vector U_i, and each question j has parameters V_j and difficulty bias Z_j.

```
from model import JML_trainer
from util import standardize_V_Z_U_promax

Y_missing = torch.load("data/Y_matrix.pt")
train_mask, test_mask = random_mask((Y_missing != -1).float(), pct=0.8)
model_FA = JML_trainer(Y_missing, K=4, mask=train_mask, device="cuda:0", is_map=True)

V, Z, U = standardize_V_Z_U_promax(model_FA.U, model_FA.V, model_FA.Z)
```

The factor model captures the **behavioral structure** of models across benchmarks and serves as the foundation for prediction.

6.3 Stage 2 — Prediction-Powered Correctness Model

The next step learns to **predict behavioral parameters** directly from metadata and semantics, without observing responses.

Two parallel predictors are trained:

- **Item-side predictor** f_V: maps question embeddings to $(\hat{V}_j, \hat{Z}_j)$
- **Model-side predictor** f_U: maps model features to $\hat{U}_i$

These predictors allow **cold-start evaluation**, predicting new entries in the response matrix Y.

6.3.1 Predicting Item Embeddings from Question Semantics

We train a neural network to map question embeddings $E_j \in \mathbb{R}^{4096}$ to latent parameters:

$$[\hat{V}_j, \hat{Z}_j] = f_\theta(E_j)$$

```
from model import embedding_V
from torch.distributions import Bernoulli

K = 4
model_V = embedding_V(input_dim=4096, output_dim=K+1).to(device)
optimizer = torch.optim.Adam(model_V.parameters(), lr=1e-3)

for epoch in range(2000):
pred = model_V(E_train)  # [n_items, K+1]
pred_V, pred_Z = pred[:, :K], pred[:, K:]
probs = torch.sigmoid(U @ pred_V.T + pred_Z.T)
```

```
loss = -Bernoulli(probs=probs[train_mask]).log_prob(Y[train_mask].float()).mean()
optimizer.zero_grad(); loss.backward(); optimizer.step()
```

The loss minimizes the **Bernoulli log-likelihood** using fixed U from the factor model.

6.3.2 Predicting Model Embeddings from Metadata

Each model has a 24-dimensional feature vector describing its **scale, architecture, and release time.**
We fit a **linear transformation** to predict U:

$$\hat{U}_i = f_\phi(F_i) = F_i W_U$$

```
from model import embedding_U

model_U = embedding_U(input_dim=24, output_dim=K).to(device)
optimizer = torch.optim.Adam(model_U.parameters(), lr=1e-3)

for epoch in range(1000):
pred_U = model_U(F_train)
loss = (pred_U - U_train).abs().mean()
optimizer.zero_grad(); loss.backward(); optimizer.step()
```

This simple mapping encourages interpretability and stable convergence.

6.4 Stage 3 — Cold-Start Evaluation

Once we have learned both mappings, we can reconstruct correctness probabilities:

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

and evaluate on unseen rows or columns of Y.

```
from torchmetrics import AUROC
auroc = AUROC(task="binary")

P_hat = torch.sigmoid(U_hat @ V_hat.T + Z_hat.T)
auc = auroc(P_hat[test_mask].cpu(), Y[test_mask].cpu())
print(f"AUC (zero-shot): {auc.item():.3f}")
```

Typical results:

Split	AUC
randcol–randcol	0.804
randrow–randrow	0.848

These confirm that the semantic–behavioral mapping generalizes well.

6.5 Mapping Semantic to Behavioral Space

To study whether semantically similar questions behave similarly,
we compute **cosine similarity** of question embeddings and **tetrachoric correlation** of their responses.

```
from util import tetrachoric_matrix_torch
import seaborn as sns, matplotlib.pyplot as plt

R = tetrachoric_matrix_torch(Y)
cosine = torch.corrcoef(V.T)

sns.scatterplot(x=cosine.flatten(), y=R.flatten(), s=5, alpha=0.5)
plt.xlabel("Cosine Similarity (semantic)")
plt.ylabel("Tetrachoric Correlation (behavior)")
plt.title("Semantic vs Behavioral Similarity")
```

Observation: Even highly similar questions (cosine > 0.99) exhibit nearly random behavioral correlations (-1 to $+1$),
showing that semantic proximity is a *poor instrument* for behavioral prediction.

6.6 Iterative Filtering via Tetrachoric Correlation

We remove inconsistent or adversarial items via **iterative filtering**.

```
from tqdm import trange

Y_filtered = Y.clone()
for t in trange(19):
R = tetrachoric_matrix_torch(Y_filtered)
p_neg = (R < 0).float().mean(1)
bad_items = torch.topk(p_neg, 500).indices
mask = torch.ones(Y_filtered.shape[1], dtype=bool)
mask[bad_items] = False
Y_filtered = Y_filtered[:, mask]
```

After 19 rounds:

- **Retained**: 11,243 of 20,743 questions (~54%)
- **Negative correlations**: ↓ from 23% → 1.67%
- **Benchmark composition**: stable across iterations

This step improves inter-item consistency and downstream factor modeling.

6.7 Generalization to New Models

We evaluate generalization to unseen models under the **randrow–randrow** split.

Predict U_{test} from metadata and evaluate:

```
U_pred = model_U(F_test)
P_hat = torch.sigmoid(U_pred @ V_frozen.T + Z_frozen.T)
auc = auroc(P_hat[test_idtor[test_row,:]].cpu(),
 ↪  Y[test_row,:][test_idtor[test_row,:]].cpu())
print(f"randrow-randrow AUC: {auc.item():.3f}")
```

Result:
AUC □ 0.8483 with $K = 1$, confirming strong linear predictability of model behavior from simple metadata.

6.8 Summary of the Prediction-Powered Framework

Component	Input	Output	Purpose
Factor model	Response matrix Y	U, V, Z	Extract latent behavior
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 judging

This pipeline allows **reliable, low-cost, and bias-resistant** measurement of model performance under cold-start conditions.

6.9 Implications

- **Efficiency**: Predicts correctness for new benchmarks without running any model queries.
- **Reliability**: Invariant to stylistic confounds.
- **Scalability**: Cost scales as $O(N + M)$ instead of $O(NM)$.
- **Interpretability**: Latent factors preserve behavioral semantics for explainable evaluation.

This tutorial is based on the paper "Measuring Without Judging: Prediction-Powered Cold-Start Evaluation" (Anonymous, 2025).
It demonstrates how factor models, semantic mapping, and adaptive filtering jointly enable a new paradigm of scalable AI evaluation.

7 Conclusion

References

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*.

Index