

Strict Graders and Hallucinated Mastery: A Cognitive Diagnostic Study of Language Models on Dynamic Programming

Sartaj Solaiman, Md Fahim Arefin*

University of Dhaka, Bangladesh

Abstract

Automated feedback in programming education typically grades correctness alone, hiding which reasoning skills a student has mastered. We apply Cognitive Diagnostic Modeling to evaluate language models as graders of Dynamic Programming, a domain that requires several reasoning skills at once. We build a five-skill Q-matrix for 28 AtCoder DP problems, with just over a quarter drawn from 2024-or-later contests to reduce contamination risk. Two large language models and three smaller models grade 560 human submissions; we fit a probe-level Cognitive Diagnostic Model (DINA) to estimate the probabilities of missing a present skill (Slip) or incorrectly crediting an absent one (Guess). The graders split sharply along a Slip-Guess axis: LLMs are strict, while smaller models often mark missing skills as present. Granite-4.1-8B is the exception, ranking third overall at our main hallucination-penalty setting. A secondary analysis of SLM-generated solutions shows a reasoning-synthesis asymmetry: the same models that recognize correct reasoning cannot produce it. CDM gives a practical diagnostic view for language-model-based grading in education, surfacing failure modes that pass@k cannot.

Keywords

Language Models, Cognitive Diagnostic Modeling, Dynamic Programming, Q-Matrix, DINA, Grader Reliability, Programming Education

1. Introduction

Language models have made code generation routine. They are also strong at introductory programming problems. They are not yet consistent on harder algorithmic problems, and Dynamic Programming (DP) is a clear example. Current evaluation of language-model code is mostly black-box. The dominant metric, pass@k, reports the fraction of problems for which at least one of k sampled completions passes all hidden tests. Such aggregate numbers tell you whether a model solved a problem, but not how or why it failed when it didn't. A timeout is also not the same failure as a wrong answer: one signals a missing optimization step, the other a broken recurrence, and a tutor should treat them differently.

A DP solution can break in several distinct ways: a poorly defined recursive state, a misread of where the subproblems overlap, a recurrence that is correct in shape but wrong at the boundary, or a working solution that times out because the optimization step is missing. Existing benchmarks do not separate these failure modes; an outcome of “wrong answer” looks the same in all of them. What we lack is a diagnostic frame that maps these deficiencies onto the cognitive skills the model is actually being asked to use.

1.1. Motivation

DP is hard for students for a reason. It is abstract, the recurrence rarely matches what an iterative thinker writes first, and a correct answer requires several reasoning skills to land at the same time. For AI to play a useful role in CS education, producing code that passes tests is not enough; the system also has to reason about *why* a particular code is right or wrong. Two questions follow:

CSEDM'26: 10th Educational Data Mining in Computer Science Education (CSEDM) Workshop, June 28, 2026, Seoul, South Korea

*Corresponding author.

✉ sartajsolaiman@gmail.com (S. Solaiman); fahim@cse.du.ac.bd (M. F. Arefin)

🌐 <https://farefin.com> (M. F. Arefin)

🆔 0009-0000-9759-2142 (S. Solaiman); 0000-0002-3413-4774 (M. F. Arefin)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

- Do current LLMs and SLMs reason about algorithmic problems, or only pattern-match against memorized solutions?
- If they reason, can we measure how far that reasoning extends?

Two specific gaps in current evaluation make these questions hard to answer.

1.1.1. The Reasoning-Synthesis Gap

Even small language models can recognize correct algorithmic solutions reasonably well [1], but that recognition does not carry over when an SLM must write the solution itself, with failures clustering around base cases, state representation, recurrence design, and edge cases. The educational stakes are concrete: an AI tutor that grades code but cannot produce a worked solution is only a partial tool. Even so, an asymmetric model remains valuable, since a high-verification, low-synthesis SLM can serve as a localized, low-latency tool that flags conceptual errors for the learner. We use DP as a stress test for this gap because its skills are well-defined and its failure modes easy to label.

1.1.2. Diagnostic Scarcity in AI Evaluation

We address this diagnostic gap with Cognitive Diagnostic Modeling (CDM), specifically the DINA (Deterministic Input, Noisy “And” Gate) model. Instead of a single accuracy number, the output is a per-skill profile for each grader and synthesizer, showing where the model is reliable and where it is not.

1.2. Contribution

This work sits at the intersection of psychometrics and AI evaluation. The key contributions are:

- **DP Q-Matrix:** a curated mapping of 28 DP problems across 5 categories (Linear DP, Grid/Tree DP, Subset DP, Probabilistic DP, Optimized DP), with each problem profiled against 5 latent skills: Problem Decomposition, State Representation, Transition Logic, Boundary Conditions, and Space/Time Optimization.
- **The DINA-AI Diagnostic Framework:** an Expectation-Maximization based probe-level DINA implementation that estimates skill mastery for LLMs and SLMs, accounting for “slipping” and “guessing” parameters at the skill-probe level.
- **Discovery of the Reasoning-Synthesis Gap:** a skill-level contrast between differentiated grader mastery profiles and floor-level synthesizer fits.
- **Grader Reliability Analysis:** we introduce the Grader Reliability Index (GRI) to evaluate the trustworthiness of LLM- and SLM-graded responses, showing that while large models tend to grade strictly, smaller models are prone to hallucinated mastery.

2. Related Work

Our work sits at the intersection of three lines of research: language-model evaluation for code, the generation-verification gap, and cognitive diagnostic modeling in education.

Evaluating LLMs on programming tasks. The dominant paradigm for code-generating language models remains outcome-based: pass@k on HumanEval, MBPP, and competitive-programming benchmarks treats a submission as binary success or failure, conflating failures that arise for entirely different reasons. LiveCodeBench [2] addresses contamination by drawing AtCoder, LeetCode, and Codeforces problems from post-cutoff dates, a strategy we follow in our problem selection. Wang et al. [3] go further, using language models to extract knowledge components from a curated knowledge base, opening a path to diagnostic rather than purely outcome-based evaluation.

LLM-as-a-judge for code. A growing literature treats language models as graders; Li et al. [4] survey its design space and reliability concerns. Recent work shows LLM judges inherit biases such as

self-preference [5] and surface-form bias on code [6], where stylistic features dominate the judgement. Our strict-grader profile is consistent with this: high Slip partly reflects stylistic strictness rather than pure logical error, and our framework quantifies that effect at the skill level.

Generation-verification asymmetry. A parallel line of work shows language models are systematically better at verifying solutions than producing them: Song et al. [7] formalize this as a limit on self-improvement, and Saad-Falcon et al. [8] show even weak verifiers can narrow the gap. This work largely treats the asymmetry on natural-language tasks; we extend the pattern into structured algorithmic reasoning and quantify it at the skill level for DP.

Cognitive Diagnostic Modeling in programming education. Cognitive diagnostic models connect item responses to latent skill profiles [9]; the DINA model performs item-level estimation from a Q-matrix of skill requirements [10]. CDM has been applied to programming education to profile student skill mastery [11], and knowledge-component discovery from code has been explored via KC-Finder [12] and code-aware knowledge tracing [13]. Recent work couples CDMs with neural representations [14], though such hybrids typically diagnose human learners. We instead treat the language models themselves as test takers, using a probe-level DINA estimator with a manually curated Q-matrix to surface their failure modes.

3. Methodology

The pipeline covers four stages: data preparation, Q-matrix construction, model fitting, and evaluation.

3.1. Data Preparation and Q-Matrix Design

This subsection covers the three preparation steps that feed into DINA: selecting a balanced and contamination-resistant problem set, encoding the skill requirements as a Q-matrix, and assembling the human-written solutions that will be graded. Each step is described below.

3.1.1. Problem Selection

Our problem set comes from AtCoder. We picked 28 DP problems spanning a range of complexity, from basic state transitions to state-compression tasks. To reduce contamination risk, 8 of the 28 problems (about 29%) are drawn from contests held in 2024 or later. This follows the time-segmented contamination-mitigation strategy used by LiveCodeBench [2]. Each problem was assigned to one of five DP families:

- Linear DP
- Grid/Tree DP
- Subset DP
- Probabilistic DP
- Optimized DP

We report results on the full 28-problem set rather than on the post-2024 subset alone. With 8 post-cutoff problems (80 AC / 80 WA submissions), a separate DINA fit on that slice is underpowered and would not support stable Slip/Guess estimates. We therefore use the recent problems to dilute contamination across the pooled set, and flag a contamination-controlled re-fit on a larger post-cutoff sample as future work.

3.1.2. Q-Matrix Formulation

We define the relation between problems and the latent skills required to solve them via a Q-matrix: a binary matrix mapping items to skills. Formally, $\mathbf{Q} \in \{0, 1\}^{J \times K}$, where $J = 28$ (DP problems) and $K = 5$ (latent skills). An entry $q_{jk} = 1$ indicates that mastery of skill k is required to produce a correct solution to problem j . The five latent skills are:

- K1 (Problem Decomposition): Identifying subproblems.
- K2 (State Representation): Defining the DP table dimensions.
- K3 (Transition Logic): Formulating the recurrence relation.
- K4 (Boundary Conditions): Correctly identifying base cases.
- K5 (Space/Time Optimization): Applying prefix sums or data structures to improve efficiency.

The Q-matrix was built by hand. To avoid a flat Q-matrix, we set $q_{jk} = 1$ whenever skill k is a target skill for problem j , not just a passively relevant one. Table 1 shows the row used for Knapsack 2. K_2

Table 1

Example Q-matrix row for Knapsack 2 before dimensionality expansion.

Problem	K1	K2	K3	K4	K5
Knapsack 2	0	1	1	0	0

and K_3 are the key skills needed for Knapsack 2. The Q-matrix is the rubric a grader is meant to follow item by item.

3.1.3. Dimensionality Alignment Expansion

To align the Q-matrix with the response representation used by the DINA estimator, each original 1×5 problem row is expanded into a 5×5 skill-probe submatrix. For each skill position k , row k in the submatrix contains a single 1 in column k if the original problem required that skill; otherwise row k is all zeros. For example, the Knapsack 2 row $[0, 1, 1, 0, 0]$ from Table 1 becomes:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Repeating this expansion for all 28 problems gives an expanded Q-matrix of size 140×5 .

3.1.4. Operational Skill-Presence Rubric

The response matrices require an operational definition of whether a skill is present in a solution. We marked a skill as present only when the solution contained a functional implementation of that reasoning step, not merely a variable name, comment, or attempted but inconsistent fragment. K_1 was present when the code decomposed the problem into reusable subproblems; K_2 when the state representation captured all information needed for the recurrence; K_3 when the transition logic matched the chosen state and problem constraints; K_4 when base cases and boundary behavior were handled correctly; and K_5 when the required optimization was actually implemented well enough to avoid the intended time or memory failure. Borderline cases were resolved conservatively as 0. For the filtered GPT-5.5 matrix, a biased grade means a positive skill mark that violated this rubric; filtering only changed 1s to 0s and never upgraded 0s to 1s.

3.1.5. Solution Gathering

From AtCoder, we collected 20 human-written solutions for each problem in the curated set, yielding 560 solutions across the 28 problems. All solutions were written in Python to keep grading conditions consistent across items. For each problem, 10 solutions had received an Accepted (AC) verdict and 10 had received a Wrong Answer (WA) verdict, keeping the graded response matrix balanced. For problems where K_5 (Space/Time Optimization) is a target skill, the WA pool was supplemented with solutions judged Time Limit Exceeded or Memory Limit Exceeded (TLE/MLE), so that each skill is

exercised by both passing and failing examples. To evaluate submissions without aggregation, each of the 560 submissions occupies one row of the response matrix, and the 140 columns are the probe items of the expanded Q-matrix (28 problems $\times$ 5 skill probes). A submission belongs to exactly one problem, so only the 5 probe columns of its parent problem are active in that row; the remaining 135 columns are masked and excluded from that row’s likelihood / scored as structural zeros. This yields a 560×140 response matrix that preserves each submission’s per-skill error profile. We fit DINA independently to one such matrix per response source — the two LLM graders, the two derived LLM matrices (Consensus, GPT-5.5_Filtered), and the three SLM graders — so the Slip, Guess, and per-skill values reported in Table 2 are per-matrix quantities.

3.2. Experimental Group Design

We split the evaluation into three groups so we can separate grading behavior from synthesis behavior:

- **Group A (LLM as graders):** We utilized two LLMs, Claude Sonnet 4.6 and GPT-5.5, to act as automated solution graders. We also report two derived response matrices. The first is GPT-5.5_Filtered, a post-hoc conservative audit of the GPT-5.5 matrix using the rubric above. The second is Consensus, a lower-bound conservative matrix: if both LLMs agreed on grading a particular skill as 1, we graded that skill 1; otherwise the response was graded 0 for that probe.
- **Group B (SLM as graders):** This group consists of three SLMs: Llama-3.1-8B, Granite-4.1-8B, and Phi-4-mini-3.8B. This serves as a benchmark for how grading reliability scales with model size.
- **Group C (SLM as synthesizers):** The same three SLMs are prompted to produce solutions for all 28 problems, creating 84 SLM-synthesized solutions (28 solutions per model). The solutions are then manually graded. Comparing Group B vs Group C allows us to quantify the Reasoning-Synthesis Gap.

3.3. Response Matrix Generation

This is the primary input for the DINA model. We employed two protocols for generating the matrix given the experimental group design:

- **Protocol A (Groups A and B):** For the graders, we parsed the model evaluation of the human-written code and mapped the grades to the Q-matrix rubric. If the model correctly identified the presence of the required skill, the probe was marked 1. In case of a conflict, the probe was marked 0 to maintain conservative estimation.
- **Protocol B (Group C):** For the synthesizers, each solution was tested against the AtCoder test suite and then manually graded with the same operational rubric. On failure, we traced the logic of the code; failure to produce correct code for a required skill resulted in 0 for that skill. For correct code, the required skill blocks were marked 1. Borderline synthesis cases were resolved conservatively as 0.

3.4. The Cognitive Diagnostic Model (DINA)

We employ the **Deterministic Input, Noisy “And” Gate (DINA)** model as our analytical engine, but our expanded Q-matrix changes the interpretation from full problem-level conjunction to skill-probe estimation. Each non-zero expanded row has a single active skill, so the ideal response for that row reduces to whether the grader has mastered that skill. We therefore interpret the fitted Slip and Guess parameters as probe-level diagnostic noise, not as estimates of a complete problem-solution success probability. This narrows the claim accordingly: the results diagnose recognition behavior over rubric probes, not full solution correctness. We first state the model formally and then describe how we fit its parameters via Expectation-Maximization.

3.4.1. Definition

The probability of grader i succeeding on item j is defined as:

$$P(X_{ij} = 1 \mid \alpha_i) = g_j^{1-\eta_{ij}} (1 - s_j)^{\eta_{ij}}$$

where α_i is the latent skill-mastery profile of grader i ; $\eta_{ij} = \prod_{k=1}^K \alpha_{ik}^{q_{jk}}$ is the ideal response (1 if grader i has mastered every skill required by item j , and 0 otherwise); s_j (Slip) is the probability that a grader fails item j despite mastering all required skills; and g_j (Guess) is the probability that a grader passes item j without mastering the required skills.

3.4.2. Fitting via Expectation-Maximization (EM)

We fit the model by EM over the $2^K = 32$ possible mastery profiles:

1. **Initialization:** mastery probabilities are initialized at 0.5, and starting values are set for s_j and g_j .
2. **E-step:** we compute the posterior that grader i belongs to profile α_l , given its response pattern $\mathbf{X}_i$:

$$P(\alpha_l \mid \mathbf{X}_i) = \frac{L(\mathbf{X}_i \mid \alpha_l) P(\alpha_l)}{\sum_{v=1}^{2^K} L(\mathbf{X}_i \mid \alpha_v) P(\alpha_v)},$$

where L is computed from the DINA item PDF over the 140 probe columns of the 560×140 matrix. Because five probes can originate from the same underlying problem and solution, we treat the resulting estimates as descriptive diagnostic summaries rather than independent item-level measurements.

3. **M-step:** we update s_j and g_j to maximize the expected log-likelihood under the current posterior.
4. **Convergence:** iteration continues until the change in marginal log-likelihood falls below 1×10^{-5} .

3.5. Evaluation Metrics

Beyond the mastery probabilities, we introduce the following evaluation metrics:

1. **Grader Reliability Index (GRI):** the GRI is defined as

$$\text{GRI} = (1 - S)(1 - G)^k,$$

where S is the Slip rate, G is the Guess rate, and k is the hallucination penalty. Our main setting uses $k = 5$, reflecting the judgement that hallucinating mastery (high G) is a more dangerous failure mode in an educational grader than over-strict grading (high S). Because this exponent is a design choice rather than an estimated parameter, we also report a sensitivity check over several k values. Higher GRI indicates a more reliable grader under the chosen penalty.

2. **Latent profile correlation:** the pairwise correlation matrix of the recovered skill profiles, indicating which skills tend to be diagnosed together by graders.
3. **Reasoning-Synthesis Delta (Δ):** the per-skill difference between mastery as a grader and mastery as a synthesizer (comparing Groups B and C). Since Group C converges to the EM floor, we interpret Δ descriptively rather than as a precise synthesizer skill estimate.

Figure 1 illustrates the full methodological pipeline. The pipeline begins with problem and solution collection, maps each problem into the Q-matrix rubric, converts model judgements into response matrices, and then fits the probe-level DINA model for evaluation.

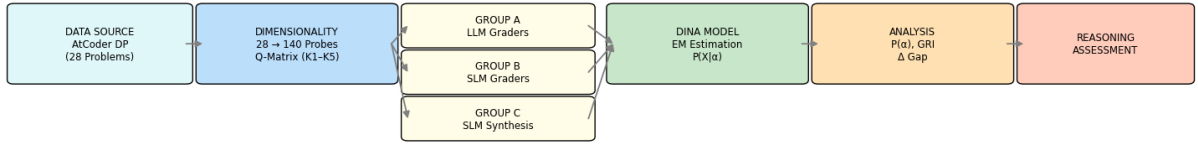


Figure 1: Research methodology: DP Diagnostic pipeline.

4. Results

We report skill mastery, grader reliability, and skill correlations.

4.1. Mastery Probability Distributions

Table 2 lists the mean Slip (S), mean Guess (G), and per-skill mastery probabilities (K_1 – K_5) for each response matrix. For Group C, the fitted synthesizer posteriors converged to the numerical floor of the EM routine, so we report them as floor-level values rather than as distinct skill estimates.

Table 2

Mastery probabilities and DINA parameters across model groups. Group C synthesizer mastery values are reported as EM-floor values because the fitted posteriors were numerically indistinguishable at the lower bound.

Group	Model	S	G	K1	K2	K3	K4	K5
GROUP A (LLM GRADERS)	Claude	0.797	0.089	0.176	0.319	0.439	0.430	0.168
GROUP A (LLM GRADERS)	GPT-5.5	0.379	0.299	0.620	0.610	0.514	0.521	0.506
GROUP A (LLM GRADERS)	Consensus	0.815	0.073	0.155	0.312	0.437	0.429	0.141
GROUP A (LLM GRADERS)	GPT-5.5_Filtered	0.803	0.135	0.176	0.372	0.338	0.179	0.180
GROUP B (SLM GRADERS)	Granite_Grader	0.624	0.212	0.209	0.209	0.209	0.137	0.543
GROUP B (SLM GRADERS)	Llama_Grader	0.786	0.196	0.183	0.381	0.352	0.156	0.243
GROUP B (SLM GRADERS)	Phi_Grader	0.062	0.414	0.471	0.471	0.471	0.471	0.471
GROUP C (SLM CODES)	Granite_Code	0.010	0.010	≈ 0 (EM floor)				
GROUP C (SLM CODES)	Llama_Code	0.010	0.010	≈ 0 (EM floor)				
GROUP C (SLM CODES)	Phi_Code	0.010	0.072	≈ 0 (EM floor)				

Probability Analysis

- The Slip-Guess Duality:** the strict Group A variants (Claude, Consensus, GPT-5.5_Filtered) pair low Guess ($G \approx 0.07$ – 0.14) with high Slip ($S \approx 0.80$), the signature of conservative grading, while raw GPT-5.5 is the outlier at $G = 0.299$, which is why we report its filtered matrix separately. High Slip should be read as a mixture of missed recognition and stylistic over-strictness.
- Group A vs. Group B:** LLMs outperform SLMs overall, especially on K_3 and K_4 , yet the strict LLM variants drop on K_1 , so identifying subproblems remains hard across the scale range tested.
- The Synthesis Floor:** Group C posteriors converge to the EM floor, indicating the model cannot recover a differentiated synthesizer profile rather than meaningful 10^{-50} -scale mastery.
- Phi-4-mini’s Non-Informative Profile:** Phi-4-mini’s identical values across K_1 – K_5 with high Guess ($G = 0.414$) point to a broad affirmative response policy rather than genuine uniform mastery.

Before psychometric analysis, raw grading performance was evaluated to establish an anchor, as shown in Table 3. The error profile maps directly to our parameter patterns. For instance, Claude’s high accuracy rate is a direct result of lack of False Positives (FP), but an elevated False Negative (FN) rate. This validates the high Slip parameter. Conversely, Phi Grader’s drop in accuracy is a direct result of aggressive over-grading (FP = 43.6%) paired with zero False Negatives. This validates the high Guess

parameter. Llama and Granite demonstrate the lowest raw error rate, before entering the DINA model optimization pipeline.

Table 3

Raw Baseline Grading Performance Matrix (percentages of 2,800 graded skill probes: 560 submissions $\times$ 5 probes)

Grader Configuration	Raw Acc (%)	FP (%)	FN (%)	TN (%)	Total Errors (%)
Claude Sonnet 4.6	91.2%	0.0%	8.8%	78.6%	8.8%
GPT-5.5	62.2%	31.7%	6.1%	46.9%	37.8%
Granite_Grader	95.1%	4.2%	0.8%	74.3%	4.9%
Llama_Grader	98.4%	0.04%	1.5%	78.5%	1.6%
Phi_Grader	56.4%	43.6%	0.0%	35.0%	43.6%

4.2. Grader Reliability

Using the Slip (S) and Guess (G) produced after fitting DINA, we ranked the graders by their trustworthiness, as shown in Figure 2a.

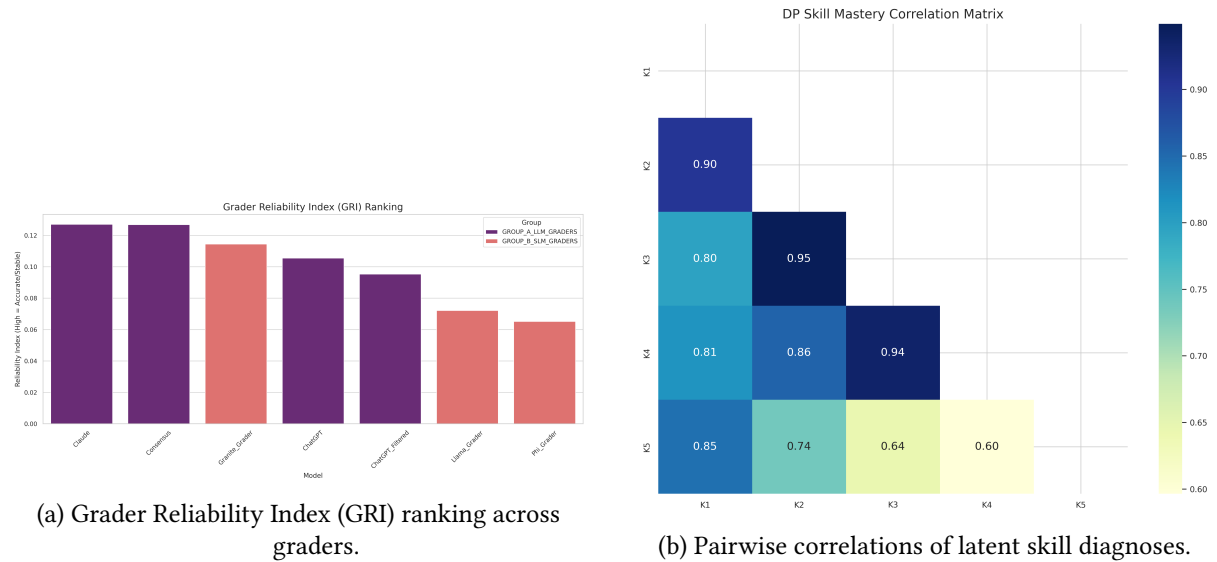


Figure 2: Comparison of grader reliability and latent skill correlations.

At the main setting $k = 5$, Claude and Consensus land at the top, and Granite-4.1-8B ranks ahead of the raw GPT-5.5 and GPT-5.5_Filtered matrices. A sensitivity check over $k \in \{1, 3, 5, 7\}$ shows that the ranking depends on how strongly hallucinated mastery is penalized: Phi-4-mini looks artificially strong at $k = 1$ because of low Slip, but drops sharply as k increases because of high Guess. We therefore make the narrower claim that, in our sample, the Slip-Guess profile separates graders more clearly than parameter count when hallucinated mastery is penalized.

4.3. Latent Skill Correlations

Figure 2b shows the correlation matrix of the latent skill diagnoses across graders.

The strongest correlation is between K_2 (State Representation) and K_3 (Transition Logic): graders that diagnose one tend to diagnose the other. K_4 (Boundary Conditions) and K_5 (Space/Time Optimization) show little coupling, so there is no analogous shortcut for the harder ends of the rubric.

4.4. The Reasoning-Synthesis Gap

For each SLM, we compare its recovered mastery as a grader with its floor-level fitted posterior as a synthesizer: $\Delta = P(\alpha_{\text{Grader}}) - P(\alpha_{\text{Coder}})$. Because Group C converges to the EM floor, Δ is descriptive rather than an independently precise synthesizer estimate; numerically, it is approximately the corresponding Group B grader mastery in Table 2. The key point is that the grading-side profile is differentiated across skills, while the synthesis-side fit provides no recoverable skill profile under the same probe representation.

5. Discussion

The central pattern is a reasoning-synthesis asymmetry: the SLMs show non-trivial grader-side mastery but collapse to floor-level fitted posteriors as synthesizers. The stronger claim is the contrast between differentiated recognition and undifferentiated synthesis, not the exact numerical value of the floor-level posteriors. For deployment, this means an SLM may be useful as a structured grader before it can produce worked DP solutions.

Because we used a strict zero-shot format, this synthesis floor reflects a limitation of standard inference rather than an absolute boundary; Chain-of-Thought or multi-turn self-correction should be tested before confirming a permanent flaw. Under current hardware constraints we suggest deploying SLMs strictly as graders rather than synthesizers, applying multi-model consensus to suppress high Guess, and delegating high-complexity skills (K_1 , K_5) to humans while the SLM focuses on state and transition logic (K_2 , K_3).

The Slip-Guess profile also matters: strict Group A variants show high Slip and low Guess, while Phi-4-mini shows low Slip and high Guess, consistent with marking missing skills as present. We therefore treat high Slip as an upper bound on grader error. In our sample this profile separates graders more clearly than parameter count, suggesting educational systems should target weak skills such as K_1 and K_5 rather than optimize only for pass/fail accuracy.

The two derived matrices should be read as audits, not independent graders: Consensus marks a skill present only when both LLMs agree, so its low Guess is partly mechanical, while GPT-5.5_Filtered tests whether removing visibly biased positives shifts the profile stricter. Keeping both rows makes the tradeoff visible, as stricter matrices reduce hallucinated mastery but risk penalizing acceptable alternate solutions. The probe-level matrix is itself equivalent to a multi-label record of which skills a grader marked present; its added value is the noisy-channel layer DINA places on top, separating a grader's latent skill tendency from Slip and Guess. This is what distinguishes over-strict grading (high Slip) from hallucinated mastery (high Guess) on the same labels, and what the GRI weights: a plain multi-label readout reports the marks, while the DINA layer reports the behavior that produced them.

5.1. Limitations

Our expanded Q-matrix turns problems into skill probes, so the DINA fit estimates skill-specific diagnostic behavior rather than full problem-level conjunction; probes from the same solution are also not fully independent. The 28-problem AtCoder set is modest, the 10 AC / 10 WA balance differs from real classrooms, and we did not semantically de-duplicate near-copy submissions. Manual synthesis labels should be read as rubric-coded outcomes rather than population estimates because we did not run a separate inter-rater reliability study. Finally, the synthesis failures are measured under one prompting setup, so prompt sensitivity and clustered uncertainty estimates remain future work.

Each item was graded in a single pass, so our Slip and Guess estimates fold in any run-to-run sampling variance. We did not repeat grading to measure intra-model stability (e.g. variance or Cohen's κ across repeated judgements of the same item), so the reported parameters should be read as single-draw estimates. Quantifying grading stability under repeated sampling is left to future work.

6. Conclusion

Probe-level CDM separates strict graders from hallucinating ones and recognition from generation. Across two LLM graders, two derived LLM response matrices, three SLM graders, and 560 graded human submissions, we find that conservative large-model matrices cluster into a high-Slip/low-Guess profile, Granite-4.1-8B remains competitive under the main GRI setting, and SLM synthesis collapses to the EM floor despite non-trivial grader-side mastery. These distinctions are invisible to pass@k. Next steps are to extend the Q-matrix beyond DP, compare probe-level and problem-level DINA fits, and test whether diagnostic feedback improves student outcomes.

A. Prompting and Model Configuration

All graders (Groups A and B) and synthesizers (Group C) were run zero-shot with no in-context examples. Decoding used a temperature of 0.1 and a maximum generation length of 5000 tokens. The same configuration was applied to every model across the three groups.

For both tasks the model returned a structured JSON object, which was parsed to the per-probe response matrix described in Section 3.3. Outputs that conflicted with the rubric or failed to parse were scored 0, consistent with the conservative-estimation policy in Protocols A and B.

A.1. Grading Prompt (Groups A and B)

Each grader received the problem identifier, the per-problem rubric entries K1–K5 drawn from the Q-matrix, and the human-written submission to be graded. The placeholders {at_id}, {rubric.get('k#')}, and {row['Solution Code']} were substituted per item at run time.

```
Task: Grade this HUMAN-written Python solution.
Problem ID: {at_id}

Rubric Requirements (1 if present/correct, 0 if missing/wrong):
- K1 (Problem Decomposition): {rubric.get('k1')}
- K2 (State Representation): {rubric.get('k2')}
- K3 (Transition Logic): {rubric.get('k3')}
- K4 (Boundary Conditions): {rubric.get('k4')}
- K5 (Space/Time Optimization): {rubric.get('k5')}

Human Code to Grade: {row['Solution Code']}

Return ONLY a JSON object:
{
  "AtCoder_ID": "{at_id}", "K1_Score": 0 or 1, "K2_Score": 0 or 1, "K3_Score": 0 or 1, "K4_Score":
0 or 1, "K5_Score": 0 or 1
}
```

A.2. Synthesis Prompt (Group C)

The three SLMs acting as synthesizers received the following system prompt. The per-problem statement and its Q-matrix targets (K1–K5) were appended to this instruction at run time. The prompt asks the model to emit k1–k5 keys alongside its solution, but these self-reported marks were discarded: only the solution_code field was retained. Each synthesized solution was executed against the AtCoder test suite and then graded by hand with the same operational rubric (Section 3.1.4), so all Group C skill scores are human-assigned and never taken from the model’s own output.

```
You are an expert competitive programmer. Solve the provided Dynamic
Programming problem. You will be given the specific Q-Matrix requirements
(K1-K5) for the problem. Your code must demonstrate mastery of the required
skills. Output STRICTLY a JSON object with the following keys: 'atcoder_id',
'problem_name', 'target_skill', 'k1', 'k2', 'k3', 'k4', 'k5', 'solution_code'.
```

Declaration on Generative AI

The language models studied in this paper (Claude Sonnet 4.6, GPT-5.5, Llama-3.1-8B, Granite-4.1-8B, Phi-4-mini-3.8B) are the *subjects* of the experiments and generate Group A and Group B grading responses and Group C synthesized solutions. Generative AI assistants were also used for copy editing and language polishing; all research contributions, claims, analyses, and conclusions are the authors' own and were reviewed by the authors before inclusion.

References

- [1] B. Liu, S. Bubeck, R. Eldan, J. Kulkarni, Y. t. Li, A. Nguyen, R. Ward, Y. Zhang, Tinygsm: Achieving >80% on gsm8k with small language models, arXiv preprint arXiv:2312.09241 (2023). URL: <https://arxiv.org/abs/2312.09241>.
- [2] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, I. Stoica, Livecodebench: Holistic and contamination-free evaluation of large language models for code, arXiv preprint arXiv:2403.07974 (2024). URL: <https://arxiv.org/abs/2403.07974>.
- [3] C. Wang, J. Lin, K. R. Koedinger, Leveraging large language models for identifying knowledge components, arXiv preprint arXiv:2511.09935 (2025). URL: <https://arxiv.org/abs/2511.09935>.
- [4] H. Li, Q. Dong, J. Chen, H. Su, Y. Zhou, Q. Ai, Z. Ye, Y. Liu, Llms-as-judges: A comprehensive survey on llm-based evaluation methods, arXiv preprint arXiv:2412.05579 (2024). URL: <https://arxiv.org/abs/2412.05579>.
- [5] K. Wataoka, T. Takahashi, R. Ri, Self-preference bias in llm-as-a-judge, arXiv preprint arXiv:2410.21819 (2024). URL: <https://arxiv.org/abs/2410.21819>.
- [6] J. Moon, Y. Hwang, D. Lee, T. Kang, Y. Kim, K. Jung, Don't judge code by its cover: Exploring biases in llm judges for code evaluation, arXiv preprint arXiv:2505.16222 (2025). URL: <https://arxiv.org/abs/2505.16222>.
- [7] Y. Song, H. Zhang, C. Eisenach, S. M. Kakade, D. Foster, U. Ghai, Mind the gap: Examining the self-improvement capabilities of large language models, in: International Conference on Learning Representations (ICLR), 2025. URL: <https://openreview.net/forum?id=Hgv0tqgaiA>.
- [8] J. Saad-Falcon, E. K. Buchanan, M. F. Chen, T.-H. Huang, B. McLaughlin, T. Bhathal, S. Zhu, B. Athiwaratkun, F. Sala, S. Linderman, et al., Shrinking the generation-verification gap with weak verifiers, arXiv preprint arXiv:2506.18203 (2025). URL: <https://arxiv.org/abs/2506.18203>.
- [9] B. W. Junker, K. Sijtsma, Cognitive assessment models with few assumptions, and connections with nonparametric item response theory, Applied Psychological Measurement 25 (2001) 258–272. doi:10.1177/01466210122032064.
- [10] J. de la Torre, The generalized dina model framework, Psychometrika 76 (2011) 179–199. doi:10.1007/s11336-011-9207-7.
- [11] M. B. Garcia, M. Tian, M. Chen, Profiling the skill mastery of introductory programming students: A cognitive diagnostic modeling approach, Education and Information Technologies (2024). doi:10.1007/s10639-024-13039-6.
- [12] Y. Shi, T. Schmucker, M. Chi, T. Barnes, T. Price, Kc-finder: Automated knowledge component discovery for programming problems, in: Proceedings of the 16th International Conference on Educational Data Mining (EDM), 2023. URL: <https://educationaldatamining.org/edm2023/proceedings/>.
- [13] Y. Shi, M. Chi, T. Barnes, T. Price, Code-dkt: A code-based knowledge tracing model for programming tasks, in: Proceedings of the 15th International Conference on Educational Data Mining (EDM), 2022. URL: <https://arxiv.org/abs/2206.03545>.
- [14] X. Chen, J. Zhang, T. Zhou, F. Zhang, Llm-cdm: A large language model enhanced cognitive diagnosis for intelligent education, IEEE Access 13 (2025) 47165–47180. doi:10.1109/ACCESS.2025.3549309.