Generative-Evaluative Agreement: A Necessary Validity Criterion for LLM-Enabled Adaptive Assessment

Grandee Lee and Yue Wang and Che Yee Lye and Luke Peh
Singapore University of Social Sciences,
463 Clementi Rd, 599494, Singapore
{grandeelee,wangyue,cylye,lukepehlc}@suss.edu.sg

Abstract

When the same LLM generates assessment items, simulates student responses, and scores them, the validation loop is self-referential. We introduce **Generative-Evaluative Agreement (GEA)**, a validity criterion measuring whether an LLM's scoring function recovers the skill levels its generative function was instructed to produce. In the first direct measurement of GEA on a two-stage adaptive assessment, the model recovers roughly half the intended variance ($r = 0.698$) with systematic positive bias. GEA is strong ($r > 0.7$) for syntactically verifiable skills but near zero for design-level skills, and low-skill overestimation inflates scores near the routing threshold. We argue that granular, skill-decomposed rubrics are the principal proposed mechanism for strengthening GEA and outline complementary mitigations.

1 Introduction

Computerized adaptive testing (CAT) traditionally relies on item banks pre-calibrated via Item Response Theory (IRT), where every item has known difficulty and discrimination parameters estimated from hundreds of real responses (van der Linden and Glas, 2010). LLM-enabled adaptive assessment disrupts this paradigm: items are generated dynamically, each student potentially receives a unique test, and classical calibration (requiring 50–200 respondents per item; Lord 1980) becomes infeasible. Revisions to rubrics, prompts, or course materials become psychometrically consequential when they change what is being measured, how performance is elicited, how responses are scored, or how scores are interpreted. In these cases, prior calibration and validity may no longer be transportable, and at least part of the item bank may need to be re-authored, relinked, re-calibrated, or revalidated (Han and Guo, 2011). In a typical school setting, learning outcomes are updated when curricula evolve, rubrics are refined each semester as instructors identify ambiguities, and course restructuring changes the skill prerequisites for each assignment. LLM-based systems absorb these changes through prompt and rubric updates alone, but the validity of the resulting assessment must be re-established each time.

This creates a **bootstrapping problem**: the system cannot be validated without real student data, but cannot be deployed at scale without prior validation. Human review of each generated item is infeasible when the item space is effectively infinite, and the classical validity pipeline (pre-calibrate, validate against human raters, then deploy) does not apply. Simulation-based validation offers a pragmatic alternative. Liu et al. (2024) demonstrated that ensembles of LLM-simulated respondents can approximate human item calibration with correlations exceeding 0.89. Zheng et al. (2026) used Monte Carlo simulation to identify optimal CAT configurations before empirical evaluation. Marquez-Carpintero et al. (2025) reviewed LLM-simulated student profiles for pre-deployment testing of pedagogical systems. However, when the same LLM generates items, simulates student responses, and scores them, the validation loop is self-referential. If the model's representation of skill levels is inconsistent across its generative and evaluative functions, the system validates itself against a distorted mirror.

This paper introduces **Generative-Evaluative Agreement (GEA)** as the formal criterion for this internal consistency: when an LLM generates a response at an intended skill level, does scoring recover that level? Valid routing decisions require scores that faithfully reflect the intended construct, but "intended difficulty" exists only in the model's internal representation, accessed through two different computational paths (generation and evaluation) that traverse different prompt-conditioned regions of the same model. Empirical verification of

Proceedings of the 21st Workshop on Innovative Use of NLP for Building Educational Applications, pages 798–812
July 3-4, 2026 ©2026 Association for Computational Linguistics

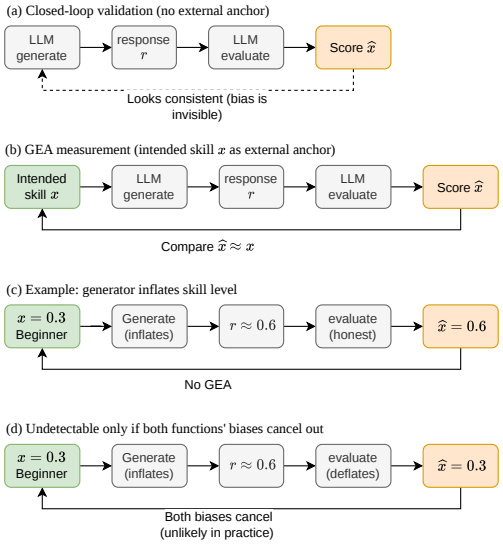

Figure 1: GEA measurement versus closed-loop self-validation. (a) In a pure closed loop, there is no external anchor and bias is invisible. (b) GEA introduces the intended skill level x as an external reference point. (c) When the generator inflates skill, GEA detects the discrepancy. (d) Bias is undetectable only if both functions share the exact same misconception, an unlikely scenario given that generation and evaluation traverse different prompt-conditioned paths.

their alignment is therefore a necessary (though not sufficient) validity condition for any LLM-based adaptive assessment that uses simulation for calibration.

1.1 Definition

Generative-Evaluative Agreement (GEA) is the degree to which an LLM's generative representation of skill levels is consistent with its evaluative representation. Formally, if the model generates a response $\mathbf{r}$ conditioned on skill level x, then scoring $\mathbf{r}$ should recover x within acceptable error bounds:

$$\mathbb{E}[\text{score}(\mathbf{r}) \mid \mathbf{r} \sim \text{generate}(x)] \approx x \qquad (1)$$

Here $x, \text{score}(\mathbf{r}) \in [0, 1]$ are continuous per-skill scores; ordinal proficiency bands (Appendix D) are derived post hoc for reporting.

We operationalise "$\approx$" through two primary metrics: Pearson r for rank-order fidelity and signed bias for systematic directionality. We propose two actionable benchmarks: $r > 0.7$ (strong GEA) to support fine-grained proficiency reporting, and $r > 0.4$ (moderate GEA) to support binary routing decisions. Skills below $r = 0.4$ should not be used for adaptive routing without human validation.

Crucially, GEA measurement is *not* equivalent to closed-loop self-validation. Figure 1 illustrates the distinction. In a pure closed-loop system (panel a), the model generates and scores with no external reference, so any systematic bias is invisible. In GEA measurement (panel b), the intended skill level x serves as an external anchor. If the generator inflates skill (panel c), the discrepancy $\hat{x} \neq x$ reveals the generation bias. GEA can only fail to detect bias when both functions share the *exact same* misconception of x (panel d), which is unlikely given the empirical evidence of divergence reviewed in Section 2.2. Figure 2 shows the concrete assessment architecture in which GEA is measured.

2 Background

2.1 The Closed-Loop Problem

In traditional CAT, item parameters and scoring functions are independently validated against real human response data. In LLM-based adaptive systems, the model performs both roles with no external anchor. An instructive (though imperfect) analogy comes from the generative/discriminative distinction: Ng and Jordan (2001) showed that models learning $P(X \mid Y)$ and $P(Y \mid X)$ can disagree at finite capacity. In an LLM, both generation and evaluation share the same weights, but are conditioned on different prompts that traverse different computational paths. Generation is dominated by fluency priors; evaluation by criterion matching. Shared architecture makes alignment *plausible* but does not *guarantee* it (Oh et al., 2024; West et al., 2023).

2.2 Empirical Evidence of Divergence

LLMs struggle to simulate lower-proficiency cognitive states (Yuan et al., 2026): expert knowledge leaks through despite skill-level prompting. Srivatsa et al. (2025) tested 11 LLMs against real NAEP data and found no model-prompt pair faithfully reproduced real student distributions; Wu et al. (2025) confirmed this for Python programming. LLMs also systematically rate their own outputs higher than equivalent text from other sources (**self-preference bias**; Panickssery et al. 2024), with the mechanism identified as perplexity-based familiarity (Wataoka et al., 2024). Even proprietary models show low intra-rater consistency at temperature > 0 (Lee et al., 2024b). In simulation-based calibration, these mechanisms compound: the result may appear internally consistent but is not exter-

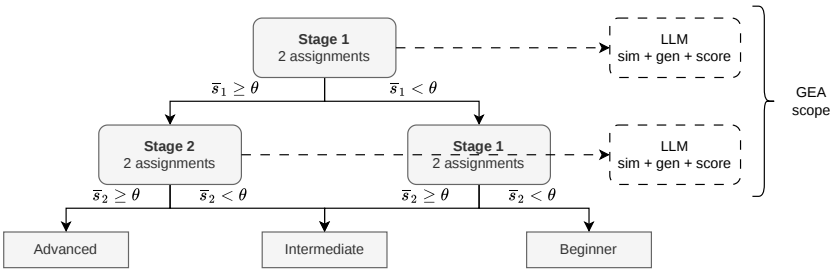

Figure 2: Adaptive assessment flow. The LLM generates assignments and scores responses at each stage. Routing depends on cumulative score $\bar{s}$ crossing threshold θ. GEA measures consistency between the LLM's generative and evaluative functions.

nally valid.

2.3 Implications for Calibration

When simulation is the only feasible calibration method, GEA becomes the gatekeeper for trustworthiness. If GEA is low, score distributions reflect the model's self-consistency rather than real student performance. Wang et al. (2025a) argue that Generalizability Theory and Many-Facet Rasch Measurement are needed to decompose multiple simultaneous error sources rather than collapsing them into a single coefficient. Even in real deployment, the generative side affects question generation: if items are at the wrong difficulty, routing decisions are based on mis-targeted items regardless of grading accuracy.

2.4 Related Work

GEA connects to several research threads. The automated essay scoring (AES) literature has studied inter-rater reliability for decades (Ramesh and Dash, 2022); GEA differs in that rater and author are the same model. From a psychometric perspective, GEA instantiates the *substantive* component of Messick's (1989) construct validity framework, and aligns with the *Standards for Educational and Psychological Testing* (American Educational Research Association et al., 2014) requirement for evidence that scores support intended interpretations. The LLM-as-judge paradigm (Zheng et al., 2024) has documented self-preference and position bias; GEA extends this from evaluation-only settings to the generate-then-evaluate pipeline where generation bias compounds with evaluation bias.

3 Empirical Measurement of GEA

This section presents the empirical measurement of GEA in the sense defined in Section 1.1 for the concrete case of Python object-oriented programming (OOP) coding tasks. The same Claude model performs both code generation and rubric-based evaluation against the 24-skill taxonomy (class definition, inheritance, exception handling, etc.; full list in Appendix B).

3.1 Simulation Design

Student profiles. We generated 150 synthetic student profiles, each comprising a 24-dimensional skill vector $\mathbf{x} \in [0, 1]^{24}$ corresponding to the official learning outcomes. (Table 2). Skills are grouped into four progressive groups: Group A (S01–S08, class basics), Group B (S01–S04, S06–S07, S09–S13, class variables and composition), Group C (S01–S04, S06, S09, S14–S21, inheritance and polymorphism), and Group D (S01, S14–S15, S22–S24, exception handling). For more details, see Appendix B. Profiles were sampled from 10 archetypes (e.g., "Absolute Beginner," "Lab 2 Proficient," "Advanced") with Gaussian noise ($\sigma = 0.04$) to produce realistic within-archetype variation.

Assessment slots. Every student attempted all 6 assignment slots regardless of skill level, bypassing the adaptive routing to ensure full coverage: Each slot tests a designated subset of the 24 skills; non-applicable skills are marked -1.0 in the output vector and excluded from scoring. Scenario entities were assigned deterministically per student (seeded on student ID) to ensure reproducibility.

Generate–then–score protocol. For each (student, slot) pair, we executed two sequential API calls to Claude Sonnet 4.6:

1. **Generate**: Given the student's full skill profile (24 skill scores with per-skill natural-language descriptors) and the assignment question, the model was prompted to produce

Table 1: Overall GEA statistics (7,788 paired skill observations, 150 students, 23 skills). 95% bootstrap CIs from 1,000 resamples.

Metric	Value	95% CI
Pearson r (pooled)	0.698	[.684, .712]
Mean bias (obs − true)	+0.059	[+.053, +.066]
Exact proficiency match	34.8%	

Python code that *precisely matches* the specified skill levels, including deliberate errors, omissions, and partial implementations for low-scoring skills.

2. **Score**: The generated code was submitted to the same model's scoring function with the identical rubric, which returned a 24-element observed skill vector $\hat{\mathbf{x}} \in \{-1.0\} \cup [0, 1]^{24}$ and a scalar score $s = \text{round}(\text{mean}(\hat{x}_i : \hat{x}_i \neq -1) \times 100)$.

This yields paired observations $(x_i, \hat{x}_i)$ for every skill i that is applicable in a given slot, providing the raw material for measuring Equation 1.

Scale. All 150 students had completed all 6 slots producing 862 result records and 7,788 paired (true, observed) skill-level observations across 23 of 24 skills (S13, Dictionary Collection Management, was not tested in any slot). Table 1 summarises the aggregate agreement statistics.

4 GEA Findings

The pooled Pearson correlation of $r = 0.698$ indicates that the LLM's evaluative function recovers roughly half the variance ($R^2 \approx 0.49$) in the true skill levels it was asked to generate. The positive mean bias of $+0.059$ confirms the direction predicted by self-preference bias: the model systematically overestimates the skill level of its own generated code.

At the proficiency-level granularity used for reporting (8 ordinal levels from *Not Demonstrated* to *Mastered*; boundaries in Appendix D), exact classification accuracy is only 34.8%, rising to 64.4% within ± 1 adjacent level. This represents moderate agreement, sufficient to distinguish broad skill bands but insufficient for fine-grained proficiency reporting.

4.1 Per-Skill GEA

GEA varies dramatically across skills. Table 2 presents the full per-skill breakdown, sorted by Pearson r. After Benjamini-Hochberg correction for multiple comparisons across 23 skills, 19 correlations remain significant at $\alpha = 0.05$; the four non-significant skills are precisely the near-zero GEA tier. Three tiers emerge:

Table 2: Per-skill GEA metrics, sorted by Pearson r. Skills marked with ⋆ are mandatory in the assessment rubric.

Skill		n	r	Bias
S05	Setter w/ Validation	290	.88	+.10
S09⋆	Class Variable	146	.83	+.26
S23	Raise Exception	145	.80	+.12
S11	Composition	139	.80	+.13
S12	Delegation	139	.80	+.17
S10	Class Var Mod	146	.78	+.14
S18⋆	Override Refine	285	.77	+.13
S24	Try/Except	145	.77	+.11
S07	Compute Method	429	.72	+.04
S15	super().__init__	433	.71	+.08
S06	__str__	717	.65	−.04
S01	Class Def	848	.62	+.06
S04	Getter Property	716	.59	+.02
S16	Subclass Attrs	288	.57	+.21
S08	Mutate Method	287	.55	−.06
S22	Custom Exception	145	.55	+.21
S02	Constructor	717	.51	−.04
S14	Subclass Def	433	.47	+.25
S03⋆	Private Vars	717	.43	±.00
S19	ABC Definition	114	.08	−.10
S17⋆	Override Replace	288	.06	+.19
S21	Polymorphism	113	−.03	−.07
S20	Concrete Subclass	108	n/a	−.09

Strong GEA ($r > 0.7$; 10 skills). Skills with concrete, syntactically verifiable indicators, such as setter validation logic (S05, $r = 0.88$), class variable declaration with underscore convention (S09, $r = 0.83$), exception raising (S23, $r = 0.80$), and composition via object attributes (S11, $r = 0.80$), show the strongest agreement. These skills have unambiguous code signatures: a `@attr.setter` with a conditional check, a `_count` class variable, a `raise CustomError()` statement. The rubric criteria map directly to syntactic patterns that both the generator and evaluator can reliably target.

Moderate GEA ($0.4 < r < 0.7$; 9 skills). Foundational skills tested across many slots (S01, S02, S04, S06) show moderate agreement ($r = 0.43$–0.65). These skills are near-universal in student code (almost every submission defines a class, writes a constructor, uses properties), creating a ceiling effect that compresses variance. Subclass-related skills (S14, S16) show moderate r but high bias ($+0.21$ to $+0.25$), indicating the generator systematically over-performs on inheritance tasks relative to the intended skill level.

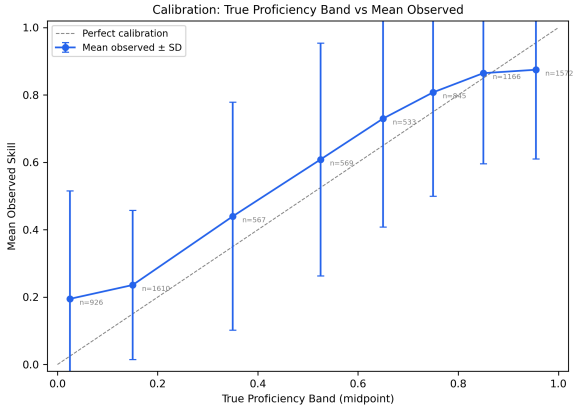

Figure 3: Calibration curve: mean observed skill as a function of true proficiency band. Error bars show ± 1 SD. The dashed diagonal represents perfect calibration. The curve lies above the diagonal at low skill levels (overestimation) and converges at high levels.

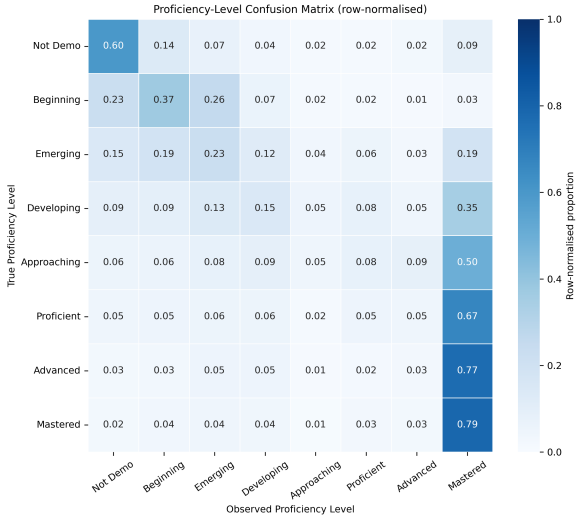

Figure 4: Row-normalised confusion matrix mapping true proficiency levels (rows) to observed proficiency levels (columns). The strong rightward bias in the "Approaching" through "Mastered" rows indicates systematic upward misclassification at higher skill levels.

Near-zero GEA ($r < 0.1$; 4 skills). Abstract design skills, including ABC definition (S19, $r = 0.08$), method overriding by replacement (S17, $r = 0.06$), polymorphism (S21, $r = -0.03$), and concrete subclass implementation (S20, constant output), show essentially no correlation between intended and observed skill levels. S20 is degenerate: the evaluator assigned nearly identical scores regardless of the true skill level, collapsing all variation. These skills require *design decisions* (choosing to define an ABC, choosing to use polymorphic dispatch) rather than *syntactic patterns*, making them harder for the generator to "partially implement" and harder for the evaluator to grade on a continuous scale.

This pattern (strong GEA for syntactically verifiable skills, weak GEA for design-level skills) is consistent with the scope limitation noted in Section 6: code assessment benefits from partially verifiable ground truth, but that benefit is concentrated in the syntactic stratum of the skill taxonomy. Design-level skills behave more like the subjective assessment domains where GEA is expected to be weakest.

4.2 Calibration and Bias Structure

The calibration curve (Figure 3) reveals an asymmetric bias:

- **Low-skill overestimation**: Students in the "Not Demonstrated" band (true ≈ 0.0) receive mean observed scores of ≈ 0.20, a $+0.20$ bias. The LLM struggles to generate authentically incompetent code; even when instructed to omit a skill, the generated code tends to include partial or vestigial implementations that the evaluator detects.

- **High-skill convergence**: Students in the "Advanced" and "Mastered" bands (true > 0.80) are scored close to their true levels, with the curve approaching the diagonal. The model finds it easier to generate competent code and to recognise competence.

- **Mid-range compression**: The "Developing" through "Proficient" bands (0.45–0.80) show the highest variance (largest error bars), suggesting the model has difficulty maintaining fine-grained distinctions in the middle of the skill range.

This asymmetry has direct implications for adaptive routing. The system's routing threshold (≥ 50 for High path) operates in the mid-range where calibration is poorest. The upward bias at low skill levels means that weak students are systematically overscored, making them more likely to exceed the threshold and be routed to the High path, exactly the misrouting scenario described in Section 6.

4.3 Proficiency-Level Classification

The confusion matrix (Figure 4) reveals the proficiency-level consequences of the biases identified above.

Upward collapse to "Mastered." The dominant pattern is a strong rightward shift: students at "Approaching" (true 0.60–0.70) are classified as "Mastered" 50% of the time; "Proficient" (true 0.70–0.80) collapses to "Mastered" 67% of the time; "Advanced" (true 0.80–0.90) reaches 77%. The evaluator effectively treats any competent implementation as "Mastered," failing to distinguish gradations within the upper half of the skill range. This is a direct manifestation of **self-preference bias**: the model's own generated code, even when deliberately degraded, retains low perplexity from the evaluator's perspective, inflating scores toward the ceiling.

Low-skill recovery. The "Not Demonstrated" level is the best-recovered category at 60% exact match, consistent with the observation that the absence of a skill (e.g., no `@property` defined at all) is a binary, syntactically verifiable condition. However, even here, 14% of truly absent skills are scored as "Beginning" and 9% as "Mastered," the latter likely reflecting cases where the generator failed to suppress the skill despite being instructed to (competence paradox leakage).

Implications for proficiency reporting. The 8-level proficiency scale is not reliably recoverable through the generate-then-score pipeline. A coarser 3- or 4-level scale would better match the system's effective resolution. For adaptive routing (threshold at 50), the upward bias at low skill levels ($+0.059 \times 100 \approx 6$ points) inflates Stage 1 scores for weak students, increasing misrouting probability to the High path.

5 Strengthening GEA

The empirical findings above reveal that GEA is partial and skill-dependent. This section examines the mechanisms available to strengthen it, beginning with the most impactful (granular rubrics) and progressing to complementary strategies that address failure modes rubrics alone cannot resolve.

5.1 Rubrics as Constrained Reasoning Paths

Section 2.1 identified the core architectural problem: generation and evaluation traverse different computational paths through the model with no guarantee of alignment. Granular rubrics address this by providing a **shared external specification** that constrains both paths to pass through the same intermediate representation, namely the rubric's criteria.

Wang et al. (2025b) formalise this as a graph reasoning problem. Without rubrics, the path from task description to score can traverse arbitrary reasoning states. With rubrics, both generation ("produce code demonstrating these specific skills") and evaluation ("check for these specific skills") are forced through the same intermediate checkpoints. Under the assumption that aligning these intermediate nodes increases the overlap between the model's generative and evaluative reasoning paths, rubric-guided assessment should exhibit higher GEA than holistic assessment.

In the context of this system, the rubric defines per-assignment skill vector tables specifying exactly which of the 24 skills are applicable to each assignment and what constitutes mastery. This provides structural decomposition: both the question generator and the response evaluator reference the same granular criteria, reducing the degrees of freedom available for the two paths to diverge. The three-tier GEA pattern in Table 2 provides indirect evidence for this mechanism: skills with unambiguous syntactic rubric criteria (e.g., "code contains a `@property` decorator") show strong GEA, while skills with holistic criteria (e.g., "demonstrates appropriate use of polymorphism") show near-zero GEA.

5.2 Empirical Support from the Literature

Rubric quality directly predicts scoring accuracy. Wu et al. (2024) found a Spearman rank correlation of $\rho = 0.94$ ($p < 0.01$) between analytic rubric alignment (with human-crafted rubrics) and automated scoring accuracy. Without rubrics, accuracy was 34.8%; with human-crafted analytic rubrics, 50.4%; with LLM-generated rubrics guided by holistic rubrics, 54.6%. The near-perfect correlation between rubric quality and scoring performance establishes that the evaluative path is strongly anchored by rubric specification. For GEA, this implies that the evaluative side of the agreement can be substantially improved through rubric design alone, without changing the model.

Rubric-aligned component extraction yields consistent gains. AutoSCORE's two-agent design (first extract rubric-relevant components into a structured JSON representation, then score based on those components) improved QWK by up to 37% (Essay Set, GPT-4o) and 74% (Science, LLaMA-8B) over single-agent baselines (Wang et al., 2025b). Gains were largest on complex, multi-dimensional rubrics, which parallels this sys-

tem's 24-skill vector with per-assignment coverage tables. The framework demonstrates that decomposing evaluation into criterion-level checks before holistic scoring reduces rubric misalignment and evaluator shortcuts. This system's architecture already follows this pattern: the evaluator produces a 24-element skill vector before computing a scalar score.

Rubrics balance accuracy across proficiency levels. Lee et al. (2024a) found that Chain-of-Thought prompting combined with scoring rubrics yielded a 13.4% accuracy increase and, critically, more balanced accuracy across different proficiency categories. Without rubrics, LLMs were biased toward certain score ranges; with rubrics, scores distributed more evenly across levels. This directly addresses the calibration asymmetry observed in Figure 3: rubric-guided evaluation may reduce the systematic overestimation at low skill levels that currently inflates routing scores.

Rubrics prevent evaluator shortcuts. Wu et al. (2024) discovered that providing graded student responses (without rubrics) actually *degraded* rubric alignment; the LLM found superficial keyword shortcuts instead of following the intended reasoning chain. Analytic rubrics prevented this by requiring criterion-level assessment, forcing the model through the intended reasoning path rather than surface-level pattern matching. This finding is particularly relevant to GEA: without rubrics, the evaluator may assign high scores based on surface features (e.g., code length, presence of class definitions) rather than the specific skill indicators the generator was instructed to produce or omit.

Rubrics are therefore a **necessary but not sufficient** condition for GEA. They are the most practical strategy within a single-model system, but complete GEA assurance requires complementary measures to address failure modes that rubric design alone cannot resolve.

5.3 Design Principles for GEA-Strengthening Rubrics

Based on the findings above, rubrics that maximise GEA should:

1. **Decompose the construct into discrete, independently assessable skills**, each with a binary or ordinal mastery indicator (as in the 24-skill vector).

2. **Specify per-assignment applicability**: explicitly mark which skills are assessed and which are not applicable, so neither generation nor evaluation drifts into unintended territory.

3. **Define decision boundaries, not just level descriptions**: specify what distinguishes mastery from non-mastery for each skill, not just what each level "looks like" holistically.

4. **Use structured output formats**: require the evaluator to produce criterion-level judgements (e.g., JSON skill vectors) before aggregating to a holistic score, following AutoSCORE's component-extraction-then-scoring paradigm (Wang et al., 2025b).

5. **Avoid excessive verbosity**: structural clarity outperforms exhaustive description; long rubrics can degrade performance in some models (Yoshida, 2025).

5.4 Complementary Mitigations

While granular rubrics are the primary strategy for strengthening GEA, they cannot address all failure modes. The competence paradox (the generator producing overly competent code despite low-skill instructions) and self-preference bias (the evaluator inflating scores for model-generated text) require complementary interventions.

Cross-model evaluation. The most direct intervention is using a different model family to score responses than the one that generated them (e.g., generate with Claude, score with GPT-4). This breaks the self-preference loop whereby a model inflates scores for its own low-perplexity outputs (Panickssery et al., 2024). The perplexity-based mechanism identified by Wataoka et al. (2024) implies that cross-model evaluation should be most beneficial for the upper proficiency levels where self-preference bias is strongest (Figure 4). Where cross-model evaluation is impractical due to cost or API constraints, **multi-sample scoring** offers a partial substitute: scoring each response multiple times at non-zero temperature and flagging high-variance items for human review surfaces the stochastic inconsistency that single-pass scoring conceals (Korthals et al., 2026).

Epistemic state specification. On the generation side, the competence paradox can be mitigated by moving beyond naive role-prompting ("act as a beginner"). Yuan et al. (2026) propose using structured misconception inventories and knowledge component graphs that constrain the generative

path to produce behaviourally realistic responses. Rather than asking the model to simulate a general proficiency level, the prompt specifies which knowledge components the student has and has not acquired, which common misconceptions are active, and which error patterns should appear. This converts a vague instruction ("produce beginner code") into a concrete specification that the generator can follow more reliably, directly improving the generative side of GEA.

Real-student pilot validation. Simulation-derived thresholds should be validated against a small real-student cohort before operational use, following standard psychometric practice. Even a pilot of 10–20 students on a subset of skills would provide an external anchor against which to calibrate the simulation's bias estimates. This is particularly important for the routing threshold, where the +6-point upward bias identified in Section 4.2 may require adjustment before deployment.

6 Discussion

Threshold sensitivity. We swept the Stage 1 routing threshold θ from 30 to 70 (Table 3), re-routing 140 students with complete data (10 excluded due to incomplete Stage 2 records).

θ	Flip%	Adv%	Int%	Beg%	Mis%
30	10.7	66.4	26.4	7.1	45.0
40	7.1	43.6	45.0	11.4	41.4
50	**0.0**	**24.3**	**60.0**	**15.7**	**34.3**
60	5.7	15.7	62.1	22.1	28.6
70	18.6	8.6	56.4	35.0	18.6

Table 3: Threshold sensitivity sweep. Flip%: routing changes vs. baseline θ=50. Mis%: fraction misaligned with true archetype ability.

A **stability plateau** spans $\theta \in [45, 55]$ (<5% flips). Misclassification decreases monotonically as θ rises (45% at θ=30 to 19% at θ=70) because the positive scoring bias pushes observed scores above true ability. However, at θ=70 nearly one in five students would be reclassified. The baseline θ=50 represents a pragmatic compromise within the stability plateau.

Decomposing GEA failure. When GEA is low, the deficit could stem from the generator, the evaluator, or both. For syntactically verifiable skills (e.g., S05, $r = 0.88$), unambiguous code signatures constrain both paths. For design-level skills (e.g., S21, $r = -0.03$), the generator likely over-produces while the evaluator lacks binary markers to assess degree. Isolating each component requires human scoring of generated code or human-written code at specified skill levels.

Domain dependence. Code assessment occupies a privileged position because programming tasks have partially verifiable ground truth: a class either defines a `@property` or it does not. Subjective domains (essay argumentation, creative writing) lack these anchors, so GEA findings here likely represent an **upper bound**. The rubric-as-config architecture is domain-agnostic, but empirical GEA guarantees are domain-specific.

Model scaling. We repeated the full simulation using Haiku 4.5 with identical rubrics and profiles (Table 4).

Metric	Haiku 4.5	Sonnet 4.6
Signed bias (obs−true)	+0.31	+0.06
Pearson r (record-level)	0.64	0.92
Terminal Advanced (%)	88.0	24.1

Table 4: Model scaling comparison. Pearson r is computed at the *record level* ($n = 862$ assignment records), which averages across ∼9 skills per record and is therefore higher than the pooled skill-level $r = 0.698$ in Table 1.

Haiku inflates scores by +17.6 points on average, scoring Absolute Beginners at 44.8 (vs. Sonnet's 19.8), above the routing threshold, and assigning 88% of students to Advanced. The pooled r difference (Sonnet 0.698 vs. Haiku 0.447) is significant (Fisher $z = 23.8$, $p < 10^{-100}$). GEA is therefore **scale-dependent**: self-preference bias and the competence paradox are amplified at smaller scale.

7 Conclusion

We introduced Generative-Evaluative Agreement (GEA) as a necessary validity criterion for LLM-enabled adaptive assessment. Using 150 synthetic profiles on a Python OOP assessment with Claude Sonnet 4.6, the model recovers roughly half the intended skill variance ($r = 0.698$, 95% CI [.684, .712]) with systematic positive bias, and GEA is strongly skill-dependent: high for syntactically verifiable skills, near zero for design-level skills.

Limitations

The reported GEA estimates should be read with several scoping constraints in mind. All 150 profiles are LLM-sampled rather than drawn from real

students; authentic learner errors likely differ from simulated ones, so a pilot of 10–20 real students would provide an external anchor for the bias estimates and a check on the simulated-error distribution. We evaluate two Claude models (Sonnet 4.6, Haiku 4.5) on Python OOP code only; cross-family replication (e.g., GPT-4o, Gemini, open-weight models) and subjective domains that lack the partial verifiability of code (essays, open-ended argumentation) would test whether the strong-vs-weak-GEA stratification we observe generalises beyond this setting. The rubric-decomposition argument and complementary mitigations (cross-model scoring, epistemic-state specification, multi-sample evaluation) are supported by prior work and by the per-skill GEA contrast in Table 2, but we do not directly ablate rubric granularity or scorer identity on the same task; comparing holistic against decomposed rubrics, and same-model against cross-model scoring, is the most immediate empirical follow-up. Finally, GEA can in principle fail to detect bias when generator and evaluator share an identical distortion—for example, a sycophantic scorer that deflates an inflated generator's score to match user expectations—and precisely quantifying this residual risk requires human-scored anchor responses to decompose GEA failure between the two functions (Section 6). The intended skill level x nonetheless remains an external reference that distinguishes GEA measurement from pure closed-loop self-validation, and GEA provides a concrete, measurable criterion any LLM-based assessment system can report before deployment.

References

American Educational Research Association, American Psychological Association, and National Council on Measurement in Education. 2014. *Standards for Educational and Psychological Testing*. American Educational Research Association, Washington, DC.

K. T. Han and F. Guo. 2011. Potential impact of item parameter drift due to practice and curriculum change on item calibration in computerized adaptive testing. GMAC Research Report RR-11-02, Graduate Management Admission Council.

Luke Korthals, Emma Akrong, Gali Geller, Hannes Rosenbusch, Raoul Grasman, and Ingmar Visser. 2026. Towards reliable LLM grading through self-consistency and selective human review: Higher accuracy, less work. *Machine Learning and Knowledge Extraction*, 8(3):74.

Gyeong-Geon Lee, Ehsan Latif, Xuansheng Wu, Ninghao Liu, and Xiaoming Zhai. 2024a. Applying large language models and chain-of-thought for automatic scoring. *Preprint*, arXiv:2312.03748.

Noah Lee, Jiwoo Hong, and James Thorne. 2024b. Evaluating the consistency of LLM evaluators. *Preprint*, arXiv:2412.00543.

Yunting Liu, Shreya Bhandari, and Zachary A. Pardos. 2024. Leveraging LLM-respondents for item evaluation: a psychometric analysis. *Preprint*, arXiv:2407.10899.

Frederic M. Lord. 1980. *Applications of Item Response Theory to Practical Testing Problems*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Luis Marquez-Carpintero, Alberto Lopez-Sellers, and Miguel Cazorla. 2025. Simulating students with large language models: A review of architecture, mechanisms, and role modelling in education with generative AI. *Preprint*, arXiv:2511.06078.

Samuel Messick. 1989. Validity. In Robert L. Linn, editor, *Educational Measurement*, 3rd edition, pages 13–103. American Council on Education and Macmillan, New York.

Andrew Y. Ng and Michael I. Jordan. 2001. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems*, volume 14. MIT Press.

Juhyun Oh, Eunsu Kim, Inha Cha, and Alice Oh. 2024. The generative AI paradox on evaluation: What it can solve, it may not evaluate. *Preprint*, arXiv:2402.06204.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. 2024. LLM evaluators recognize and favor their own generations. *Preprint*, arXiv:2404.13076.

Dhananjay Ramesh and Sandeep Kumar Dash. 2022. An analysis on the state of automated essay scoring. *Preprint*, arXiv:2205.04083.

KV Aditya Srivatsa, Kaushal Kumar Maurya, and Ekaterina Kochmar. 2025. Can LLMs reliably simulate real students' abilities in mathematics and reading comprehension? *Preprint*, arXiv:2507.08232.

Wim J. van der Linden and Cees A. W. Glas. 2010. *Elements of Adaptive Testing*. Statistics for Social and Behavioral Sciences. Springer, New York.

Yuehan Wang, Jinyan Huang, Lun Du, Yuxin Guo, Ying Liu, and Rong Wang. 2025a. Evaluating large language models as raters in large-scale writing assessments: A psychometric framework for reliability and validity. *Computers and Education: Artificial Intelligence*, 9:100481.

Yun Wang, Zhaojun Ding, Xuansheng Wu, Siyue Sun, Ninghao Liu, and Xiaoming Zhai. 2025b. AutoSCORE: Enhancing automated scoring with multi-agent large language models via structured component recognition. *Preprint*, arXiv:2509.21910.

Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. 2024. Self-preference bias in LLM-as-a-judge. *Preprint*, arXiv:2410.21819.

Peter West, Ximing Lu, Nouha Dziri, Faeze Brahman, Linjie Li, Jena D. Hwang, Liwei Jiang, Jillian Fisher, Abhilasha Ravichander, Khyathi Chandu, Benjamin Newman, Pang Wei Koh, Allyson Ettinger, and Yejin Choi. 2023. The generative AI paradox: "what it can create, it may not understand". *Preprint*, arXiv:2311.00059.

Tao Wu, Jingyuan Chen, Wang Lin, Mengze Li, Yumeng Zhu, Ang Li, Kun Kuang, and Fei Wu. 2025. Embracing imperfection: Simulating students with diverse cognitive levels using LLM-based agents. *Preprint*, arXiv:2505.19997.

Xuansheng Wu, Padmaja Pravin Saraf, Gyeong-Geon Lee, Ehsan Latif, Ninghao Liu, and Xiaoming Zhai. 2024. Unveiling scoring processes: Dissecting the differences between LLMs and human graders in automatic scoring. *Preprint*, arXiv:2407.18328.

Lui Yoshida. 2025. Do we need a detailed rubric for automated essay scoring using large language models? *Preprint*, arXiv:2505.01035.

Zhihao Yuan, Yunze Xiao, Ming Li, Weihao Xuan, Richard Tong, Mona Diab, and Tom Mitchell. 2026. Towards valid student simulation with large language models. *Preprint*, arXiv:2601.05473.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2024. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *Preprint*, arXiv:2306.05685.

Tianpeng Zheng, Zhehan Jiang, Jiayi Liu, and Shicong Feng. 2026. Leveraging computerized adaptive testing for cost-effective evaluation of large language models in medical benchmarking. *Preprint*, arXiv:2603.23506.

A Assessment Architecture

The system is a conversational assessment tool delivered via Telegram Bot that conducts adaptive, scenario-based coding assessments. Students work through multiple progressive mini coding assignments per stage, evaluated by Claude AI against predefined rubrics. The system routes students to a terminal difficulty level based on cumulative stage performance.

Two-stage adaptive routing. All students begin with Stage 1 (2 assignments covering class basics and composition). Based on their cumulative Stage 1 score relative to threshold θ, students are routed to either the *High Performer* path (inheritance and exception handling) or the *Low Performer* path (reinforcement of foundational skills). After Stage 2, a terminal level is assigned:

- **Advanced**: Stage 2 High path, cumulative $\geq \theta$
- **Intermediate**: Stage 2 High path, cumulative $< \theta$; *or* Stage 2 Low path, cumulative $\geq \theta$
- **Beginner**: Stage 2 Low path, cumulative $< \theta$

Dynamic question generation. Questions are not pre-stored. Claude generates each assignment dynamically at runtime based on the rubric, the current stage/path, and a scenario template. Each student receives a slightly different variation of the same assignment—same core objective and difficulty, but different scenario entities (e.g., bank account, airline, cinema), discouraging copying. Every assignment begins with an ASCII UML class diagram showing the class(es) the student must implement.

Scoring. Each submission is evaluated by Claude against the rubric for that assignment. The model returns a 24-element skill vector $\hat{\mathbf{x}} \in \{-1.0\} \cup [0, 1]^{24}$ (where -1.0 denotes inapplicable skills) and a scalar score $s = \text{round}(\text{mean}(\hat{x}_i : \hat{x}_i \neq -1) \times 100)$. The scalar score drives routing; the skill vector provides diagnostic detail.

B Skill Taxonomy

The 24 skills are organised into four progressive groups corresponding to the course's lab sequence. Skills marked with $\star$ are mandatory anchors in the assessment rubric.

C Assessment Slot–Skill Coverage

Table 6 shows which skills are assessed (scored 0.0–1.0) in each of the 6 assessment slots. Skills not listed for a slot receive -1.0 (not applicable) in the skill vector.

D Proficiency Scale

Continuous skill scores are mapped to ordinal proficiency levels using the boundaries in Table 7. These levels are used for the confusion matrix analysis (Section 4.3) and for natural-language descriptors in student profiles.

E Student Profile Archetypes

150 synthetic student profiles were sampled from 10 archetypes (Table 8). Each archetype defines per-group skill ranges from which individual skill scores are drawn uniformly, with Gaussian noise ($\sigma = 0.04$) added to produce within-archetype variation. Profiles were generated with a fixed random seed for reproducibility.

Example profile. Table 9 shows a representative "Lab 2 Developing" profile. The student has strong class basics (Group A, mean 0.84), partial composition skills (Group B, mean 0.46), and minimal inheritance/exception knowledge (Groups C–D, < 0.10). Each skill carries a natural-language descriptor (shown for selected skills) that is included in the generation prompt to constrain the LLM's code output.

F Prompt Templates

F.1 Code Generation Prompt (Simulation)

For each (student, slot) pair, the following prompt is sent to Claude Sonnet 4.6 to generate code that matches the student's skill profile. The `STUDENT SKILL PROFILE` block lists only the skills applicable to the current slot, with each skill's numeric score, proficiency level, and natural-language descriptor.

```
You are simulating a student submitting
a Python OOP coding assignment.

The student's exact skill profile for
the skills tested in this assignment is
provided below. Write Python code that a
student at PRECISELY these skill levels
would produce. Faithfully reflect each
described weakness and strength — do not
average them out or homogenise the code.

STUDENT SKILL PROFILE (relevant skills
only):
- S01 Class Definition: 0.82 (Advanced) —
Class is well-defined, correctly named,
and structurally complete with multiple
methods.
- S03 Private Instance Variables: 0.70
(Proficient) — self.__attr used for key
attributes with property getters; not
all attributes are private.
- ...

ASSIGNMENT:
{the generated assignment text}

Rules:
1.  Output Python code only — no
explanations, no markdown fences, no
preamble.
2. For skills rated "Not Demonstrated"
or "Beginning", the code must clearly
exhibit the described gap.
3.  For skills rated "Advanced" or
"Mastered", that aspect of the code must
be correct and complete.
4. Each skill reflects its own level
independently — the code can be strong
in one area and weak in another.
5. Use realistic student-style naming
and formatting consistent with the
described skill levels.
```

F.2 Scoring Prompt (Evaluation)

The scoring function sends the full rubric document (including per-assignment skill vector tables with scoring guidance) along with the student's submission. The prompt instructs the model to:

```
You are a coding assessment scorer for
a Python OOP course.

The full rubric is below.

RUBRIC:
{full RUBRICS.md content}

—

Score the student's submission for:
- Stage: {stage}  - Path: {path}  -
Assignment: {n} of 2

Assignment given to the student:
"""{question text}"""

Student's submission:
"""{student code}"""

Instructions:
1. Locate the rubric section for this
stage, path, and assignment number.
2. Fill in the 24-element skill_vector
exactly as defined in the Skill Vector
table.
- Use -1.0 for skills marked -1.0 (not
applicable).
- Use a float 0.0–1.0 for all other
skills, following the scoring guidance.
- Use intermediate values (e.g. 0.3, 0.7)
freely.
3. Compute score = round(mean(v_i for
v_i if v_i != -1.0) * 100).
4. Write 2–4 sentences of constructive
feedback.

Return ONLY a valid JSON object:
{"score": <int>, "feedback": "<text>",
"skill_vector": [<s01>, ..., <s24>]}
```

F.3 Question Generation Prompt

The question generator receives the full rubric and is instructed to locate the correct section for the given stage, path, and assignment number, substitute the student's scenario entity into the template, produce the class diagram followed by coding instructions, and return only the assignment text the student will see. Scenario entities are drawn from pre-defined lists:

- **Stage 1**: bank account, airline booking, cinema, grade tracker, event planner, inventory, point-of-sale, membership

- **Stage 2 High**: banking hierarchy, airline, vehicle fleet, cinema chain, course catalogue, training programme
- **Stage 2 Low**: contact book, recipe manager, budget tracker, shopping list, event log, book collection

Entities are assigned deterministically per student (seeded on student ID) to ensure reproducibility across simulation restarts.

G Rubric Excerpts: Strong vs. Weak GEA Skills

To illustrate the rubric specificity hypothesis (Section 5.1), we present the scoring guidance for a strong-GEA skill and a near-zero-GEA skill as they appear in the rubric's skill vector tables.

S05 — Setter with Validation ($r = 0.88$, strong GEA). The rubric specifies concrete syntactic markers at each score level:

> **Scoring guidance**: 1.0 = `@attr.setter` with a meaningful validation rule; 0.5 = setter present but no validation logic; 0.0 = absent.

The 8-level proficiency descriptors used in student profiles further constrain generation:

The binary nature of the criterion (setter with validation present or absent) provides an unambiguous code signature that both the generator and evaluator can reliably target.

S21 — Polymorphism via Shared Interface ($r = -0.03$, near-zero GEA). The rubric criterion is holistic rather than syntactic:

> **Scoring guidance**: 1.0 = loop/function calls same method on mixed-type list without `isinstance` checks; 0.5 = loop present but uses type checks; 0.0 = no polymorphic usage.

The corresponding proficiency descriptors:

Polymorphism is a *design choice* (choosing to iterate over a heterogeneous list) rather than a syntactic pattern, making it harder for the generator to "partially implement" and harder for the evaluator to grade on a continuous scale. The descriptor levels describe *degrees of design completeness* rather than presence/absence of identifiable code tokens.

H Routing Logic

The routing state machine operates as follows. Let $\bar{s}_k$ denote the cumulative (mean) score after completing all assignments in stage k.

1. **Stage 1 completion**: Compute $\bar{s}_1$. If $\bar{s}_1 \geq \theta$, route to Stage 2 High path; otherwise Stage 2 Low path.
2. **Stage 2 completion**: Compute $\bar{s}_2$.
   - High path: if $\bar{s}_2 \geq \theta$ then *Advanced*; else *Intermediate*.
   - Low path: if $\bar{s}_2 \geq \theta$ then *Intermediate*; else *Beginner*.

The threshold $\theta = 50$ is a placeholder pending real-student calibration. The threshold sensitivity analysis (Table 3) confirms a stability plateau at $\theta \in [45, 55]$ where $<5\%$ of routing decisions change.

Each stage comprises exactly 2 assignments. Scores within a stage are averaged (not summed), so $\bar{s}_k = \frac{1}{2}\sum_{j=1}^{2} s_{k,j}$ where $s_{k,j} \in [0, 100]$. Sessions are stored in memory keyed by Telegram user ID and do not survive bot restarts.

Table 5: Complete 24-skill taxonomy with descriptions and demonstration criteria.

ID	Skill	Description	Demonstrated by
Group A — Class Basics (Lab 1)			
S01	Class Definition	Defines a class using `class` with PascalCase name; non-empty body	Any valid class definition with $\geq$1 method
S02	Constructor Init	Uses `__init__(self, ...)` to initialise instance attributes	`__init__` present, $\geq$2 attrs assigned via `self`
S03$\star$	Private Vars	Declares `self.__attr` with name mangling	$\geq$1 `self.__attr` with property/getter access
S04	Getter Property	`@property` decorator exposes private attribute	Method with `@property` returning `self.__attr`
S05	Setter w/ Validation	`@attr.setter` with $\geq$1 validation rule	Setter checks condition before assigning
S06	`__str__`	Returns human-readable string using instance data	f-string with $\geq$2 instance attributes
S07	Compute Method	Instance method reading `self` attrs, returns computed value	`return` using `self.attr` in calculation
S08	Mutate Method	Instance method modifying `self` attributes	Assigns new value to `self.__attr`
Group B — Class Variables & Composition (Lab 2)			
S09$\star$	Class Variable	Shared class-level `_attr = value` (not in `__init__`)	`_attr` at class level, accessed via `ClassName._attr`
S10	Class Var Mod	Correctly accesses/modifies class variable at runtime	Change reflected across all instances
S11	Composition	Has-a relationship: object stored as attribute (DI)	`__init__` accepts object param, assigns to `self.__other`
S12	Delegation	Outer class calls composed object's methods/properties	`self.__inner.method()` in outer class
S13	Dict Collection	Dictionary manages collection with add/search/remove	`dict[key] = obj`, `dict.get()`, `dict.pop()`
Group C — Inheritance (Lab 3)			
S14	Subclass Def	`class Child(Parent):` with correct parent	Subclass instance can call parent methods
S15	`super().__init__()`	Calls `super().__init__(...)` first in subclass	First statement in child `__init__`
S16	Subclass Attrs	New attributes added after `super()` call	$\geq$1 `self.__new_attr` after `super()`
S17$\star$	Override Replace	Completely replaces parent method (no `super()`)	Same-name method with entirely new logic
S18$\star$	Override Refine	Calls `super().method()` then extends result	`super().method()` + additional logic
S19	ABC Definition	`ABC` + `@abstractmethod` to define contract	`from abc import ABC, abstractmethod`
S20	Concrete Subclass	Implements all abstract methods with meaningful bodies	Subclass instantiates without `TypeError`
S21	Polymorphism	Same method called on mixed-type list, no type checks	Loop over heterogeneous list, no `isinstance`
Group D — Exception Handling (Lab 4)			
S22	Custom Exception	Class inheriting from `Exception`	`class MyException(Exception): pass`
S23	Raise Exception	`raise` custom exception with descriptive message	`if condition: raise MyException("msg")`
S24	Try/Except	`try/except` catching custom + built-in exceptions	$\geq$2 distinct conditions caught with messages

Table 6: Slot-to-skill mapping. Each slot tests a designated subset of the 24 skills; remaining skills are marked -1.0.

Slot	Content	Skills assessed	n skills
Stage 1, A1	Lab 1: classes & properties	S01–S08	8
Stage 1, A2	Lab 2: composition	S01–S04, S06–S07, S11–S12	8
Stage 2 High, A1	Lab 3: inheritance	S01–S04, S06, S09–S10, S14–S21	15
Stage 2 High, A2	Lab 4: exceptions	S01, S14–S15, S22–S24	6
Stage 2 Low, A1	Lab 1 reinforcement	S01–S08	8
Stage 2 Low, A2	Lab 3 introduction	S01–S04, S06, S14–S18	10

Table 7: Proficiency level boundaries.

Level	Score range	Midpoint
Not Demonstrated	$[0.00, 0.05)$	0.025
Beginning	$[0.05, 0.25)$	0.15
Emerging	$[0.25, 0.45)$	0.35
Developing	$[0.45, 0.60)$	0.525
Approaching	$[0.60, 0.70)$	0.65
Proficient	$[0.70, 0.80)$	0.75
Advanced	$[0.80, 0.90)$	0.85
Mastered	$[0.90, 1.00]$	0.95

Table 8: Archetype definitions. Each cell shows the $[\text{lo}, \text{hi}]$ range for uniform sampling of skill scores within that sub-group. Sub-groups: A = S01–S08 (class basics), B = S09–S13 (composition), C_1 = S14–S16 (basic subclass), C_2 = S17–S18 (mandatory overrides), C_3 = S19–S21 (advanced inheritance), D = S22–S24 (exceptions).

Archetype	%	A	B	C_1	C_2	C_3	D
Absolute Beginner	8	.00–.22	.00–.10	.00–.07	.00–.07	.00–.05	.00–.05
Lab 1 Developing	12	.22–.52	.00–.14	.00–.09	.00–.09	.00–.05	.00–.05
Lab 1 Proficient	15	.52–.78	.00–.22	.00–.14	.00–.12	.00–.07	.00–.05
Lab 1 Mastered	12	.78–1.0	.18–.48	.00–.18	.00–.14	.00–.09	.00–.05
Lab 2 Developing	12	.68–1.0	.32–.62	.00–.18	.00–.14	.00–.09	.00–.07
Lab 2 Proficient	10	.78–1.0	.62–.92	.08–.28	.05–.22	.00–.12	.00–.09
Lab 3 Developing	12	.72–1.0	.62–1.0	.28–.62	.22–.58	.00–.22	.00–.14
Lab 3 Proficient	10	.82–1.0	.72–1.0	.58–.88	.52–.85	.08–.38	.00–.18
Lab 4 Developing	5	.78–1.0	.72–1.0	.68–1.0	.62–.92	.18–.58	.28–.62
Advanced	4	.85–1.0	.80–1.0	.78–1.0	.75–1.0	.52–.92	.68–1.0

Table 9: Excerpt from student profile 0114 (Lab 2 Developing archetype, overall score 0.41).

Skill	Name	Score	Level
S01	Class Definition	0.82	Advanced
S03$\star$	Private Vars	0.70	Proficient
S05	Setter w/ Validation	0.92	Mastered
S09$\star$	Class Variable	0.37	Emerging
S11	Composition	0.66	Approaching
S14	Subclass Def	0.15	Beginning
S17$\star$	Override Replace	0.00	Not Dem.
S22	Custom Exception	0.01	Not Dem.

Table 10: S05 proficiency descriptors (excerpt).

Level	Descriptor
Not Dem.	No setter defined; private attributes cannot be updated after construction.
Emerging	Setter present but validation logic is absent; any value is accepted.
Proficient	Setter enforces a clear validation rule and handles the invalid case appropriately; minor gap.
Mastered	Setter enforces thorough validation with appropriate response; all edge cases covered.

Table 11: S21 proficiency descriptors (excerpt).

Level	Descriptor
Not Dem.	No polymorphic usage; objects are type-checked before any method call.
Emerging	A polymorphic call is made but only one subclass type is present.
Proficient	Multiple subclass types in a collection; same method called without type checks; each responds correctly.
Mastered	Polymorphism fully and cleanly demonstrated; shared interface called on mixed-type collection with no type checks and correct output per type.