Grading Is Easy, Feedback Is Hard: Evaluating LLMs for Design-Oriented OOP Assessment

Eli P. C. Silva
Universidade Federal de Campina Grande
Campina Grande, Brazil
eli.pedro@copin.ufcg.edu.br

Eliane C. Araújo
Universidade Federal de Campina Grande
Campina Grande, Brazil
eliane@computacao.ufcg.edu.br

Everton L. G. Alves
Universidade Federal de Campina Grande
Campina Grande, Brazil
everton@computacao.ufcg.edu.br

Abstract

Assessing Object-Oriented Programming (OOP) assignments is difficult to scale, as instructors must evaluate both functional correctness and design qualities such as class modeling, responsibility distribution, and separation of concerns. Although large language models (LLMs) can support rubric-based code assessment, there is limited empirical evidence on their ability to approximate human grading while identifying pedagogically relevant design problems. This paper presents an evaluation protocol for LLM-assisted OOP assessment that separates two dimensions: grade concordance with human assessments and diagnostic alignment with expert-annotated code problems. We instantiate the protocol in a study with Gemini 3.1 Pro and GPT-5.5 on 84 anonymized Java submissions from an introductory programming course, using the same assignment, rubric, prompt structure, and output schema. This study revealed distinct assessment behaviors that would be obscured by a single aggregate score. Gemini achieved higher grade concordance (MAE = 0.44, bias = +0.14, with 92.5% of run-level grades within ±1.0 point), whereas GPT-5.5 showed systematic undergrading (MAE = 1.48, bias = −1.45, with 27.4% of run-level grades within ±1.0 point). In diagnostic alignment, GPT-5.5 achieved a higher run-averaged recall than Gemini (0.96 vs. 0.62) but lower precision (0.52 vs. 0.69), resulting in similar F1-scores (0.67 vs. 0.65). Within this assignment and prompting protocol, the results show why LLM-based assessment should be evaluated as a human-in-the-loop workflow: grade approximation and diagnostic coverage are separable capabilities that may require different forms of calibration and instructor review.

Keywords

Automated Assessment, Large Language Models, Programming Education, Object-Oriented Programming, Object-Oriented Design

1 Introduction

In introductory Object-Oriented Programming (OOP) courses, students learn that software quality is a broad concept encompassing not only functional correctness but also sound design decisions [22, 30]. In this setting, students are expected to develop design reasoning skills, including decomposing problems into classes, assigning responsibilities, modeling object relationships, encapsulating state, and separating interface logic from domain behavior. Recent research in OOP education highlights abstraction and conceptual software modeling as fundamental learning challenges and objectives for novice students, strengthening the connection between introductory programming tasks and broader software engineering concerns such as maintainability, modularity, and extensibility [35].

This challenge becomes particularly evident in assignments where functional correctness and design quality do not fully align. A submission may correctly implement the required features but exhibit poor object modeling, high coupling, or logic concentrated in procedural structures. Conversely, another submission may contain minor implementation gaps yet still reflect sound design principles. Traditional automated assessment approaches, such as test-based grading and static analysis, are effective in identifying correctness and certain code quality issues but offer limited support for assessing responsibility distribution, object modeling, and other deeper aspects of design [2, 15, 24]. Consequently, design-oriented assessment remains largely dependent on human judgment.

This distinction is important because grades and feedback serve different functions. Grades summarize performance, while written feedback must explain errors, justify their significance, and guide improvement. In practice, teaching assistants (TAs) often grade large numbers of submissions under significant time and consistency constraints, which can lead to variable assessment practices and uneven feedback quality [5, 25]. This limitation is particularly critical in OOP assignments, where students may receive satisfactory grades without clear guidance on abstraction, responsibility distribution, or coupling issues.

Large Language Models (LLMs) have recently emerged as tools capable of analyzing source code, mapping implementations to natural-language rubrics, and generating explanatory feedback on code. Prior work has explored LLMs for a range of programming-education and code-analysis tasks [3, 7, 12, 21, 28, 34]. Among applications concerned with code assessment, recent studies have used LLMs to detect code smells [10] and test quality issues [23]; however, such approaches focus primarily on localized defects rather than holistic design assessment. These capabilities make LLMs promising for contexts where consistently producing high-quality feedback is costly, but empirical evidence on their effectiveness in assessing object-oriented design remains limited.

This lack of evidence partly reflects how automated assessment is typically evaluated. Existing approaches focus on functional correctness or overall grade concordance, with limited attention to whether systems identify the same conceptual and design issues considered important by instructors [19]. Aggregate metrics can obscure diagnostic differences, as similar scores may reflect different interpretations of student design. In this context, educational value depends not only on assigning an appropriate score but also on identifying underlying misconceptions and structural flaws.

Beyond this methodological limitation, the use of LLMs in educational evaluation raises concerns about reliability, consistency, calibration, and pedagogical alignment [8, 17, 31]. These concerns

are particularly salient in design-oriented assessment, where submissions with similar functional behavior may reflect substantially different levels of abstraction, modularization, and design quality.

This paper investigates a protocol for evaluating LLMs as instructor-support tools for rubric-based assessment of an OOP assignment in an undergraduate Computer Science course. The protocol is instantiated with Gemini 3.1 Pro[1] and GPT-5.5[2], but the goal is not to rank models or choose a universal best model for grading and formative feedback. Instead, the study uses two leading model families as contrastive cases to examine why LLM-assisted assessment may need to be evaluated along two separate dimensions: grade concordance and diagnostic accuracy. The analysis uses 84 anonymized Java submissions collected across two semesters, comparing model grades with TA-assigned grades and an expert reference assessment, and comparing model-reported problems with expert-annotated programming and design issues. In this setting, the results show that a model can be closer to human grades while missing relevant issues, or recover more issues while producing more false positives and harsher grades. These findings motivate an evaluation framework that treats score calibration and problem identification as distinct yet related aspects of assessment.

The study is guided by the following research questions:

- **RQ1: To What Extent Do Model Grades Concord With an Expert-Reference Assessment, Relative to the Operational TA Baseline, in a Design-Oriented OOP Assignment?**
- **RQ2: To What Extent Do LLMs Identify Problems in Students' Code That Are Relevant to Diagnostic Feedback?**

The contributions of this work are threefold: (i) an evaluation protocol that separates grade concordance from diagnostic alignment; (ii) empirical evidence from a design-oriented OOP assignment using authentic student submissions and an expert reference; and (iii) a workflow-oriented analysis showing how grade calibration and diagnostic coverage imply distinct human-in-the-loop uses.

2 Related Work

Automated assessment has long been used in programming education to provide scalable grading, rapid feedback, and consistent execution of predefined checks. Classical systems typically rely on unit tests, input-output comparison, static analysis, or combinations of these techniques [2, 15]. These approaches are especially effective when the expected behavior can be specified as executable tests or when violations can be expressed as syntactic or structural rules.

However, correctness-oriented automation captures only part of what instructors evaluate. Reviews of automated feedback systems show an emphasis on localized defects rather than holistic interpretation under multi-criterion rubrics [19]. This limitation is particularly relevant to OOP, where behavioral correctness can coexist with weak class modeling or responsibility assignment. Rubric-based OOP assessment can improve consistency but remains time-intensive [30], while abstraction and conceptual modeling remain central challenges for novices [35].

LLMs extend automated assessment by interpreting rubrics, reasoning about code, and generating explanations [31]. Studies of programming assistance, feedback, grading, and code review show that useful results depend on prompt design, rubric alignment, validation, and calibration; technically plausible feedback may still be generic, overly detailed, or misaligned with course level [6–8, 12, 21, 27]. Evaluations of code-smell detection, quality judgments, and OOP design likewise show useful capabilities alongside sensitivity to surface attributes and uneven reasoning about abstractions [10, 28, 33].

Feedback quality is not equivalent to grade concordance: effective feedback must connect evidence to learning goals and support subsequent action [13]. Frameworks for LLM-supported assessment therefore emphasize pedagogical integrity [1], while empirical work reports difficulty producing actionable guidance aligned with instructional priorities [20]. Limited evidence jointly evaluates rubric-based grading of authentic introductory OOP submissions and diagnostic alignment with expert-annotated design issues. This study addresses that gap by separating grade concordance from instructor-relevant diagnostic alignment.

Recent studies are complementary but differ from our evaluation target. Work on automated code-review comments examines whether models can generate plausible review observations [12], while code-smell and quality-assessment studies focus on localized technical judgments or sensitivity to code presentation [10, 28]. OODEval evaluates whether models can produce or reason about object-oriented designs [33], rather than assess heterogeneous novice solutions under an authentic course rubric. Studies of grading and formative feedback, in turn, typically emphasize score agreement or the quality of generated comments [6, 20, 27]. Our protocol connects these lines by evaluating the same complete submissions at two levels—score concordance and problem-instance alignment—and by repeating each model assessment three times while retaining the submission as the inferential unit. This combination exposes calibration, diagnostic coverage, and run-to-run variability as distinct properties of an assessment workflow.

3 Methodology

This section describes the empirical design used to evaluate LLM-assisted assessment of introductory object-oriented programming submissions. Two LLMs are compared under controlled conditions, using the same assignment, rubric, student submissions, prompt structure, and output schema. The study focuses on two complementary dimensions: (i) grade concordance with human reference assessments (RQ1) and (ii) diagnostic alignment with expert-annotated programming and design issues (RQ2). The implications for AI-assisted assessment workflows are synthesized from these two analyses.

3.1 Study Context and Design

The study was conducted in a first-year Object-Oriented Programming (OOP) course within an undergraduate Computer Science program. In addition to theoretical instruction, the course includes programming assignments designed to reinforce core OOP concepts. In these assignments, students start from a provided codebase and progressively refactor or extend it to practice concepts such as

[1] https://ai.google.dev/gemini-api/docs/models/gemini-3.1-pro-preview
[2] https://developers.openai.com/api/docs/models/gpt-5.5

class decomposition, encapsulation, object composition, collection management, equality semantics, and separation of concerns. All submissions are written in Java and managed through an online classroom platform.

Submissions are typically assessed through manual code review by teaching assistants (TAs), guided by a rubric defined by the course instructor. For this study, 84 submissions of the same assignment were collected across two semesters, each produced by a different student. These submissions were graded by 28 undergraduate TAs assigned to support the course during that period.

At the beginning of the course, all TAs participated in instructor-led orientation meetings. These meetings clarified what was expected from code review, how the official rubric should be applied, and what kind of feedback students should receive. The goal was to align review practices across TAs, emphasizing evidence-based deductions, consistency with course-level design expectations, and actionable comments for students. This shared preparation supports interpreting TA grades as an *operational baseline* that reflects regular course practice under common instructions, rather than as a fully calibrated consensus reference.

TAs performed complementary roles: some primarily accompanied students throughout the assignment, whereas others served as dedicated reviewers. Accordingly, reviewer TAs graded most submissions, while accompaniment-focused TAs typically evaluated only the submissions of the students they followed closely. Each submission was graded by one TA, with workloads ranging from 1 to 6 submissions per TA (mean = 3.0). The full submission-to-TA mapping is available in the replication package[3].Therefore, TA grades should be interpreted as operational course assessments rather than a consensus ground truth. This distinction is important because TA evaluations may reflect calibration differences, time constraints, or variations in rubric interpretation.

Although all TAs followed the same rubric and received the same orientation, the grading process remains inherently subjective and may introduce variation in calibration, specificity, and problem-level feedback across evaluators. Individual TA experience was not recorded in the artifact, and TA identities remain anonymized (`TA01`–`TA28`); therefore, TA grades are analyzed as a pooled operational baseline rather than by modeling individual TA calibration. This setting is particularly suitable for investigating LLM-assisted assessment, as it requires judgment over design decisions and code organization, rather than relying solely on functional correctness or syntactic properties [2, 19, 30].

During the course, only TA assessments were produced. LLM-based and expert reference assessments were generated post hoc using the same anonymized submissions, assignment specification, and grading rubric. Figure 1 summarizes the pipeline: Phase 1 prepares common inputs and reference artifacts; Phase 2 assembles the operational TA grades and produces post hoc expert and LLM assessments; and Phase 3 analyzes grade concordance for RQ1 and diagnostic alignment for RQ2.

TA comments were not available as consistent problem-level annotations for all submissions; consequently, an expert diagnostic reference was constructed for RQ2.

3.2 Assignment and Dataset

The selected assignment, *FilmNow*, is an OOP programming task in which students implement and refactor a movie-catalog system. The starter code has a limited structure and requires students to introduce a more appropriate object-oriented design. Expected improvements include creating domain classes, replacing primitive or string-based representations with object references, managing collections correctly, encapsulating state, implementing equality semantics where needed, and separating user-interface responsibilities from domain behavior.

The assignment was open-ended within the constraints defined by its specification and rubric. The specification described the required behaviors and provided design guidance, but did not prescribe a step-by-step refactoring process or a single solution path. Students started from the same partially implemented system and were expected to evolve key components, particularly `FilmNow` and the user-interface code, while preserving the required behavior. Although instructors and teaching assistants used the rubric during grading, it was not explicitly shared with students beforehand.

The guide encouraged the introduction of a `Filme` class and illustrated alternative design choices, while the grading rubric allowed equivalent implementations (e.g., arrays or Java collections) as long as they respected the assignment constraints. As a result, submissions varied not only in functional completeness but also in the design strategies adopted: some students introduced localized classes, others restructured responsibility boundaries more extensively, and others retained largely procedural logic with minimal structural changes. This diversity is central to the evaluation, as it requires LLMs to assess heterogeneous refactoring outcomes rather than compare solutions against a single canonical implementation.

This assignment was chosen because it foregrounds design decisions central to introductory OOP learning. In particular, students must identify appropriate classes and responsibilities, model relationships through object composition, manage collections consistently, encapsulate internal state, implement equality semantics for domain objects, and separate interface concerns from domain logic. As such, it captures common difficulties in early OOP education, particularly abstraction, class decomposition, and responsibility distribution [22, 30]. Details about the assignment are available in the replication package[4].

The dataset contains 84 anonymized Java submissions, labeled S01–S84, collected across two semesters. Each submission corresponds to a different student. Before LLM evaluation, submissions were anonymized by masking metadata lines, replacing platform-identifying mentions, and using only anonymized submission identifiers in prompts, outputs, and analyses. For each submission, the collected materials include the source code, criterion-level TA scores, TA feedback when available (although this feedback was incomplete and heterogeneous across submissions and was therefore not used as a consistent reference for diagnostic evaluation), expert reference scores and annotations, and the corresponding LLM-generated assessments.

[3] https://github.com/elipcs/grading-is-easy-feedback-is-hard/blob/main/data/lab03-filmnow/runs/gemini31pro/run1/inputs/manifests/ta_submission_assignment.csv

[4] https://github.com/elipcs/grading-is-easy-feedback-is-hard/blob/main/data/lab03-filmnow/runs/gemini31pro/run1/inputs/assignment/lab_specification.md

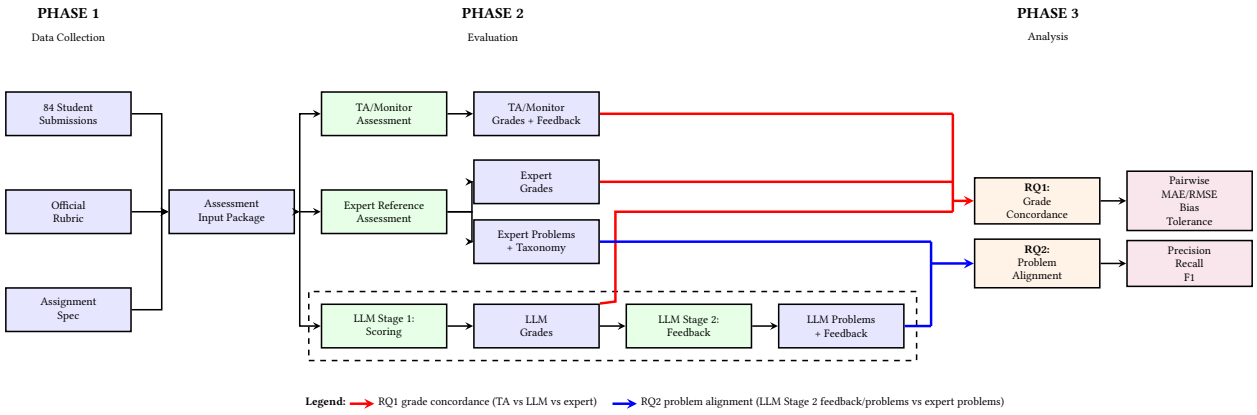

Figure 1: Experimental pipeline. Phase 1 collects anonymized student submissions, the assignment specification, the official rubric, and operational TA assessments. Phase 2 produces post hoc expert reference assessments and LLM assessments for Gemini 3.1 Pro and GPT-5.5, with both LLMs using the same two-stage prompting protocol. Phase 3 triangulates grade concordance among TA, expert, and model grades for RQ1, and compares model-reported diagnostic issues with expert-annotated issues for RQ2.

3.3 Assessment Rubric

The official course rubric contains seven criteria, each scored on a 0–10 scale and combined through weighted aggregation. Let s_i denote the score assigned to criterion i and w_i its corresponding weight, with $\sum_i w_i = 1$. The final grade is computed as follows:

$$G = \sum_{i=1}^{7} w_i s_i$$

The rubric combines the core OOP and functionality criteria, which represent 60% of the final grade, with supporting criteria, which represent 40%.

Core OOP and functionality criteria (60%):

- **Class Modeling (15%):** adequacy of class decomposition, cohesion, and responsibility distribution;
- **Basic Functionality (10%):** correctness of core features;
- **Object References (15%):** appropriate use of object references instead of primitive duplication;
- **Collections and Arrays (10%):** adequacy of collection choices and usage; and
- **Separation of Concerns (10%):** distinction between interface logic and domain logic.

Supporting criteria (40%):

- **Unit Tests (20%):** presence and quality of tests; and
- **Readability and Documentation (20%):** naming, formatting, clarity, and documentation when applicable.

3.4 LLM Assessments

The study selected Gemini 3.1 Pro (`gemini-3.1-pro-preview`) and GPT-5.5 (`gpt-5.5`), one high-capacity model from each of two commercial providers, as contrastive cases. To be included, a model had to provide programmatic API access, a context window large enough to ingest the full assignment package (specification, rubric, starter reference, and submission code), and reliable structured JSON generation under temperature 0 with high reasoning effort or thinking enabled. The goal was not to benchmark all eligible models or claim market representativeness, but to examine whether assessment behavior differs across provider families under a common protocol. Both models evaluated the same submissions under an identical two-stage prompting protocol.

Each prompt package included the assignment specification, official grading reference, starter scaffold, rubric JSON, course-level evaluation principles, abstract calibration examples, and the anonymized submission package. Stage 1 defined the model as an evaluator for an introductory OOP laboratory and required criterion-by-criterion JSON scoring. The mandatory procedure required the model to (i) identify what each criterion measures, (ii) record positive evidence, (iii) identify material failures with code evidence, (iv) separate penalizable failures from non-penalized observations, (v) assign severity and confidence to every deduction, and (vi) consolidate duplicate symptoms of the same root cause. The embedded course principles instructed the model to evaluate only rubric-supported evidence, give partial credit in first-year settings, and measure learning rather than demand professional-grade code. The reviewer-audit rules further required that optional improvements be routed to `non_penalized_observations` rather than converted into point loss, and discouraged penalties for naming, helper structure, formatting, casing, or message punctuation unless materially relevant to the rubric. Abstract calibration examples additionally prohibited cosmetic details from contaminating central criteria and limited deductions to at most three per criterion unless the rubric required otherwise.

Stage 2 converted the validated Stage 1 JSON into student-facing feedback without re-grading. It selected at most six high-impact, high-confidence deductions, omitted low-confidence items by default, consolidated repeated symptoms, and prohibited advanced techniques outside the lab scope. API settings were matched across provider clients, with temperature set to 0 and high reasoning effort

or thinking enabled. The prompts used are available in the replication package[5]. Because both models received the same prompt structure, schema, and course-level constraints, observed differences are interpreted as model-specific assessment behavior rather than protocol variation.

For each model, all 84 submissions were evaluated three independent times under the same two-stage prompting protocol, yielding 252 LLM assessments per model (504 in total). Each run used identical prompt packages, rubric inputs, anonymized submission code, and API settings (temperature 0 with high reasoning effort or thinking enabled). Although temperature 0 reduces sampling variability, it does not guarantee deterministic outputs across repeated API calls. The three runs are therefore treated as technical replications of the same assessment protocol rather than as independent observations. Throughout the analysis, the submission remains the inferential unit ($n = 84$); the 252 assessments per model characterize repeated measures on the same submissions and are not treated as an independent sample of size 252.

3.5 Expert Reference Assessments

To support both RQ1 and RQ2, expert reference assessments were constructed for all 84 submissions. These assessments were produced by a single evaluator with instructional and subject-matter expertise in OOP assessment. The evaluator independently reviewed each submission, assigned criterion-level scores using the same seven-criterion rubric, and recorded problem annotations whenever a code-level observation justified a deduction or indicated a pedagogically relevant issue. The expert annotations were produced independently from the model outputs; however, because the course grades already existed, the expert reference is not treated as a blinded or consensus adjudication.

For each identified issue, the evaluator produced a textual description, linked it to the most relevant rubric criterion, and classified it according to a 12-category taxonomy covering the topics of *class modeling*, *responsibility division*, *object-reference usage*, *collections and arrays*, *equality semantics* (`hashCode`/`equals`), *string comparison*, *List validation*, *output format*, *input handling*, *tests*, *readability*, and *other implementation concerns*. When a single code observation affected multiple rubric criteria, the evaluator associated it with the most directly impacted criterion while using the taxonomy to preserve the broader issue category. Each issue instance was counted once under its primary criterion to avoid double-counting in diagnostic-accuracy metrics. In total, the expert reference contains 361 annotated issues across the 84 submissions (mean = 4.3 issues per submission), distributed across the seven rubric criteria and twelve diagnostic categories.

The scoring component of the expert reference was used as the analytical benchmark for grade-concordance metrics, whereas the annotation component served as the reference for diagnostic alignment analysis. This reference is not treated as objective ground truth or consensus adjudication because it was produced by a single evaluator. Instead, it provides structured validity evidence for the interpretations examined in this study, consistent with the view that validity concerns the appropriateness of interpretations supported by assessment evidence rather than the existence of a uniquely correct score [26]. To improve transparency and auditability, the protocol preserved code-based evidence, criterion associations, taxonomy labels, and structured annotations.

[5]https://github.com/elipcs/grading-is-easy-feedback-is-hard/tree/main/src/prompt

3.6 Analysis Procedure

3.6.1 RQ1: Grade Concordance. For RQ1, TA grades and expert-reference grades (0–10 scale) are used in a triangulation design. TA grades represent operational course assessment, whereas the expert reference provides a consistent analytical benchmark based on the same rubric. The analysis first compares TA and expert grades to characterize the level of agreement between operational grading and expert judgment. The LLM-generated grades are then interpreted relative to this triangulated human-reference context.

To quantify grade concordance, final grades on the 0–10 scale are compared using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), median absolute error, mean signed bias, and the proportion of run-level model grades within ±1.0 point. Bias is computed as the model grade minus the expert-reference grade; negative values indicate undergrading and positive values indicate overgrading. Agreement metrics quantify numerical similarity to expert scores, but do not establish autonomous grading reliability or assessment validity. High agreement indicates consistency with the reference; validity additionally depends on whether scores reflect the intended learning construct and support appropriate educational interpretations [16, 26].

Because each submission was evaluated three times per model, grade concordance is analyzed as a repeated-measures design ($n = 84$). The MAE is estimated by first averaging the absolute errors of the three runs within each submission, and then averaging across all 84 submissions. In contrast, the RMSE is computed conventionally over the flattened set of all 252 run-level observations. Neither estimator averages the three run-level grades before computing the error, because that would represent an ensemble output and understate the variability of a single real execution. Bias and the proportion of run-level grades within ±1.0 point are likewise computed from individual run-level errors. The repeated structure is preserved for inference by resampling submissions as clusters, retaining all three observations from each selected submission.

For each model, concordance metrics are also computed separately for each of the three runs, and the mean of those run-specific metrics is reported together with the minimum–maximum range across runs to characterize sensitivity to execution. Ninety-five percent confidence intervals are obtained by submission-level bootstrap resampling, preserving the three-run structure within each submission.

Direct model comparison uses the mean absolute error averaged across three runs for each submission and model. Paired GPT–Gemini differences are tested with a paired permutation test at the submission level and summarized by the mean paired difference and its submission-level bootstrap 95% confidence interval. To position model error relative to the operational baseline, each model's run-averaged absolute error is also compared with the TA absolute error for the same submission using paired permutation tests and bootstrap confidence intervals. Score distributions across the expert reference, TA, and the two models are compared with a Friedman test on submission-matched grades, followed by pairwise

Wilcoxon signed-rank contrasts with Holm correction. For these score-distribution tests, model grades are averaged across the three technical runs within a submission; these runs are not treated as independent assessment sources but as repeated measures. A non-significant difference in absolute error is interpreted as an absence of evidence of a difference, not as a formal equivalence result.

3.6.2 RQ2: Diagnostic Alignment. For RQ2, expert annotations are used as the reference for diagnostic alignment: whether a model surfaced problems corresponding to expert-identified issues, not whether the final report was clear, motivating, or pedagogically well sequenced. Operational TA feedback was too incomplete and heterogeneous to support a consistent problem-level reference.

LLM-reported issues were compared with expert annotations at the problem-instance level. A match required semantic equivalence for the same submission and rubric criterion; taxonomy overlap alone was not sufficient [19]. The pipeline extracted structured deduction fields from the canonical JSON outputs, created `submission_id|criterion_id` comparison units, and restricted matching to issues within the same unit.

Semantic adjudication was implemented with Claude Sonnet 5 (`claude-sonnet-5`)[6] using temperature 0 and structured JSON output. The adjudicator defined matches by shared root cause or specific bug, allowed subsumption and many-to-many coverage, and returned covered, uncovered, and unmatched lists used as true positives, false negatives, and false positives, respectively. Because this step is model-assisted, the labels are treated as operational matches rather than objective ground truth. The adjudicator was selected from a different provider family than either evaluated model, which mitigates provider-family bias in matching; the same prompt, schema, and criterion-restricted procedure were applied to both model conditions without revealing the evaluated model identity. Matching prompts, model identifiers, and cache entries are included in the artifact package.

Following standard classification-evaluation definitions [29], *Precision* is the proportion of model-reported issues that match expert-reference issues, *Recall* is the proportion of expert issues identified by the model, and *F1-score* is their harmonic mean. Results are reported comparatively across models rather than against a fixed acceptance threshold, because no established cutoffs exist for LLM-based diagnostic alignment in design-oriented OOP assessment.

Because each submission was evaluated three times per model, semantic matching is executed separately for every model–submission–run combination. Precision, recall, and F1 are computed for each run, and TP, FP, and FN are reported at the run level. Run-level TP, FP, and FN are *not* pooled across runs as if they were independent observations. Descriptive model-level precision and recall are computed from aggregate TP, FP, and FN within each run and then averaged across the three runs; their 95% confidence intervals use submission-level bootstrap resampling that preserves each submission's three-run structure. Inferential comparisons between models first compute precision and recall at the submission–run level and then average across runs within each submission, retaining $n = 84$. Paired submission-level exact sign tests compare which model obtained higher run-averaged precision or recall per submission, excluding cases where precision is undefined because either model reported no issues in any run.

3.6.3 Run-to-Run Stability. Beyond RQ1 and RQ2, run-to-run stability is analyzed explicitly for model-generated grades under a fixed protocol. The central stability metrics are absolute agreement and run-pair differences rather than tests of systematic run ordering. For each model, absolute inter-run agreement for final grades is summarized with the two-way random-effects intraclass correlation coefficient ICC(2,1) and its 95% confidence interval. Run-pair MAE and RMSE quantify the typical grade shift between executions on the same submission. Exact grade agreement reports the proportion of submissions receiving identical final grades across all three runs; agreement within ±0.5 and ±1.0 points reports the proportion of submissions whose run-level grades remain within those tolerances.

A Friedman test across runs can detect systematic run-to-run shifts, but it does not by itself establish stability; ICC and absolute run-pair differences are therefore treated as the primary stability analyses.

4 Results

4.1 RQ1: To What Extent Do Model Grades Concord With an Expert-Reference Assessment, Relative to the Operational TA Baseline, in a Design-Oriented OOP Assignment?

Table 1 summarizes grade concordance against the expert reference, utilizing run-averaged absolute errors per submission ($n = 84$) and RMSE over all 252 repeated observations. Operational TA grades provide a useful human baseline: across the 84 submissions, TA–expert agreement yielded MAE = 0.45 and bias = +0.13. Against the same expert reference, Gemini 3.1 Pro achieved substantially stronger concordance than GPT-5.5. Gemini obtained a run-averaged MAE of 0.44 (95% bootstrap CI [0.35, 0.54]), RMSE = 0.67 (95% CI [0.49, 0.86]), and bias = +0.14, with per-run MAE ranging from 0.39 to 0.54. GPT-5.5 exhibited systematic undergrading: run-averaged MAE = 1.48 (95% CI [1.34, 1.62]), RMSE = 1.67 (95% CI [1.49, 1.85]), bias = −1.45, and mean grade = 7.26 versus expert mean = 8.71, with per-run MAE ranging from 1.39 to 1.60.

Figure 2 reinforces this contrast: Expert, TA, and Gemini grades are concentrated in the upper range, whereas GPT-5.5 shows a clear downward shift. Across Expert, TA, and run-averaged model grades, a Friedman test rejected equality of the score distributions ($\chi^2(3) = 154.56$, $p < .001$, Kendall's $W = 0.61$) [9, 18]. Post hoc Wilcoxon signed-rank tests with Holm correction showed that GPT-5.5 grades were significantly lower than Expert, TA, and Gemini grades (all $p_{\text{Holm}} < .001$). Both Gemini and TA grades differed from the expert reference ($p_{\text{Holm}} = .002$ for each), whereas Gemini and TA grade distributions did not differ significantly ($p_{\text{Holm}} = .365$) [14, 32].

Error-based paired comparisons answer the baseline-relative component of RQ1 more directly. GPT-5.5's run-averaged absolute error exceeded Gemini's by a mean of 1.04 points (95% bootstrap CI

[6] https://docs.anthropic.com/en/docs/about-claude/models

Table 1: Grade concordance with the expert reference assessment (84 submissions; 252 repeated observations per LLM). Bias is evaluator grade minus expert grade.

Evaluator	Mean grade	MAE	RMSE	Bias	Median AE	Within ±1	Within ±2
TA (operational baseline)	8.85	0.45	0.63	+0.13	0.38	91.7%	98.8%
Gemini 3.1 Pro	8.85	0.44	0.67	+0.14	0.30	92.5%	96.8%
GPT-5.5	7.26	1.48	1.67	−1.45	1.38	27.4%	81.0%

Note. LLM MAE averages run-level absolute errors within each submission before aggregating over submissions (n = 84); RMSE is the conventional root mean squared error over all 252 run-level observations. "Within" reports the proportion of individual run-level grades satisfying the tolerance, averaged within submission. No metric is computed from a three-run ensemble grade, and the 252 assessments per model are repeated measures rather than independent observations.

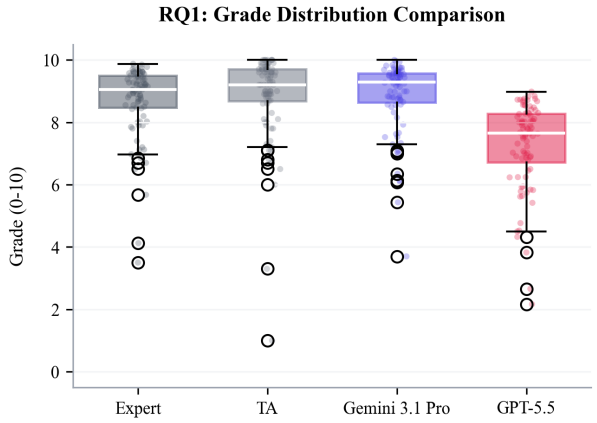

Figure 2: Distribution of expert-reference grades, TA grades, and model-generated grades. Boxes show the interquartile range with the median line; whiskers extend to the most extreme values within 1.5 × IQR, and jittered points show individual submissions, including possible outliers beyond the whiskers.

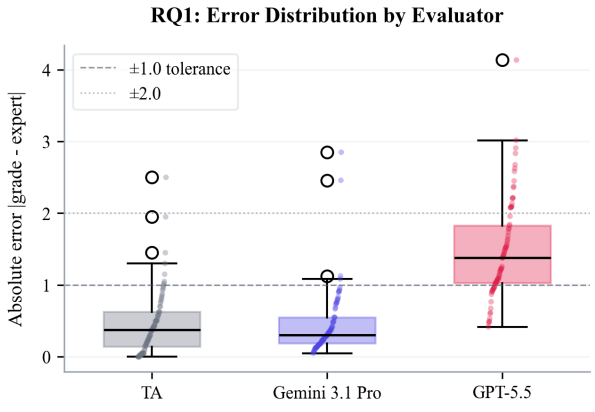

Figure 3: Distribution of absolute grade errors |model − expert| **by model.**

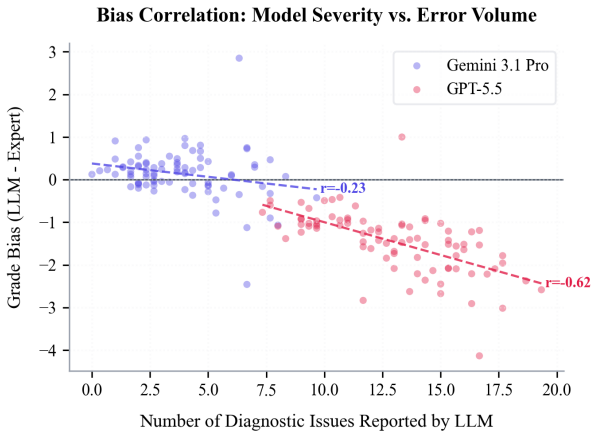

Figure 4: Relationship between reported diagnostic issues and grade bias for each model.

[0.89, 1.18]; paired permutation $p < .001$). Relative to the TA absolute error on the same submissions, Gemini's mean difference was −0.01 points (95% CI [−0.10, 0.08]; paired permutation $p = .815$), providing no evidence of a difference in concordance error, but not establishing formal equivalence. GPT-5.5's absolute error exceeded the TA baseline by 1.02 points on average (95% CI [0.87, 1.18]; paired permutation $p < .001$). Thus, score-distribution contrasts and concordance-error contrasts support complementary interpretations.

Figure 3 presents the distribution of absolute grading errors. Gemini closely approximates the TA baseline, with 92.5% of individual run-level grades within ±1.0 point of the expert reference, while GPT-5.5 exhibits a wider error distribution: only 27.4% fall within ±1.0 point, although 81.0% remain within ±2.0 points.

Figure 4 relates grade bias to the number of diagnostic issues reported by each model. At the submission level, reported issue volume was negatively associated with grade bias for both GPT-5.5 ($r = -0.55$, $n = 84$, $p < .001$) and Gemini ($r = -0.54$, $n = 84$, $p < .001$), indicating that submissions with more reported issues tended to receive lower grades relative to the expert reference. These descriptive associations do not establish that issue volume caused the observed grading behavior. The practical difference lies in the scale of issue reporting: GPT-5.5 reported a mean of 8.0 issues per submission and more often undergraded, whereas Gemini reported a mean of 3.9 issues and remained closer to the expert-reference distribution.

Run-to-run results qualify this concordance analysis. Final-grade ICC(2,1) was high for both Gemini (0.89, 95% CI [0.85, 0.92]) and

GPT-5.5 (0.90, 95% CI [0.75, 0.96]), indicating strong relative ordering despite absolute shifts. Mean run-pair MAE was 0.33 for Gemini (pairwise range 0.19–0.41) and 0.39 for GPT-5.5 (0.31–0.46). Exact agreement across all three runs was only 2.4% and 0%, respectively, although 86.9% of Gemini submissions and 89.3% of GPT-5.5 submissions remained within ±1.0 point across runs.

Answering RQ1, LLM-generated grades can align with expert and TA assessments, but this alignment depends on both the model and the instructional role assigned to its output. In the analyzed dataset, Gemini's behavior was more compatible with a grade-auditing role, whereas GPT-5.5 would require substantial recalibration before its grades could be used even as instructor-facing evidence. Grade concordance is therefore not an inherent property of LLM-assisted assessment as it emerges from the interaction among model behavior, calibration, and how reported issues are translated into deductions. This finding extends prior observations about grading variability in programming education [5, 25] by showing that AI assistance introduces an additional source of assessment variability.

4.2 RQ2: To What Extent Do LLMs Identify Problems in Students' Code That Are Relevant to Diagnostic Feedback?

The run 1 category distribution highlights three prevalent issue families. Of 78 expert-annotated missing-test occurrences, GPT-5.5 matched all 78 while adding 6 unmatched reports; Gemini matched 39 and missed 39. For class modeling, GPT-5.5 matched 55 of 58 reference occurrences but added 23 unmatched reports, whereas Gemini matched 27 and added only 3. For List validation, GPT-5.5 matched all 48 reference occurrences and added 33 reports, while Gemini matched 45 and added 19. These categories illustrate the broader contrast between exhaustive and selective reporting. The complete twelve-category distribution is available in the replication package[7].

Across the full run 1 distribution, GPT-5.5 produced 666 category-presence occurrences (346 matched, 320 unmatched, and 15 reference occurrences missed), while Gemini produced 326 (223 matched, 103 unmatched, and 138 missed) against 361 expert occurrences. Counts collapse repeated rows to binary presence for each submission–category pair and are descriptive; inferential comparisons use run-averaged submission-level precision and recall ($n = 84$). Across three runs, GPT-5.5 achieved higher run-averaged recall (0.96, 95% bootstrap CI [0.94, 0.98], range 0.96–0.96) but lower precision (0.52, 95% CI [0.49, 0.55], range 0.51–0.53), whereas Gemini achieved higher precision (0.69, 95% CI [0.66, 0.73], range 0.68–0.71) and lower recall (0.62, 95% CI [0.58, 0.66], range 0.61–0.64). Aggregate F1 was similar (0.67 for GPT-5.5 vs. 0.65 for Gemini). Paired exact sign tests on submission-level run averages showed GPT-5.5 higher recall in 78 submissions, Gemini higher in 0, with 5 ties ($p < .001$); Gemini higher precision in 73 submissions, GPT-5.5 higher in 9, and 1 tied ($p < .001$).

As a sensitivity check, each borderline run 1 matching decision changes precision by at most one divided by the number of reported issues (1/666 for GPT-5.5 and 1/326 for Gemini) and recall by at most 1/361. Thus, even a reversal of ten adjudicated matches would shift GPT-5.5 precision by about 0.015 and recall by about 0.028, and Gemini precision by about 0.031 and recall by about 0.028. This does not remove the main contrast between GPT-5.5's high-recall, low-precision profile and Gemini's more selective profile, but it reinforces that exact category-level counts should be interpreted cautiously.

These results highlight a clear trade-off. GPT-5.5 is substantially more exhaustive, recovering nearly all reference issues across runs, but its diagnostic output is noisy. Gemini is more selective and more precise, but its lower recall means that many expert-annotated issues would remain invisible if its output were used as the only diagnostic source. Aggregate run-averaged F1 was nearly identical across models (0.67 for GPT-5.5 vs. 0.65 for Gemini), but this similarity masks opposite error profiles: GPT-5.5 achieved higher recall (0.96 vs. 0.62) at the cost of lower precision (0.52 vs. 0.69).

Answering RQ2, LLMs identified expert-annotated programming and design problems with run-averaged F1-scores near 0.66–0.67, but their diagnostic performance reflected a coverage–precision trade-off. GPT-5.5 recovered nearly all reference issues (recall = 0.96) but produced many unmatched reports (precision = 0.52), often because it applied professional code-review standards beyond the introductory rubric. Gemini was more precise (0.69) but missed over one-third of reference issues (recall = 0.62). Thus, similar F1-scores mask distinct pedagogical risks: noisy candidate generation versus omission of relevant problems.

4.3 Run-to-Run Stability

Table 2 summarizes grade stability across the three technical replications per model. Final-grade ICC(2,1) was high for both models (Gemini = 0.89, 95% CI [0.85, 0.92]; GPT-5.5 = 0.90, 95% CI [0.75, 0.96]), indicating strong relative ordering of submissions across runs despite non-trivial absolute shifts. Run-pair MAE between executions was modest for Gemini (0.19–0.41 points) and somewhat larger for GPT-5.5 (0.31–0.46 points). Exact grade agreement across all three runs was low (2.4% for Gemini; 0% for GPT-5.5), but 86.9% (Gemini) and 89.3% (GPT-5.5) of submissions remained within ±1.0 point across runs.

5 Discussion

5.1 Interpretation of Findings

The central finding concerns workflow role rather than model superiority. Rubric-based assessment of OOP submissions decomposes into two related tasks: assigning scores calibrated to a course rubric and identifying the issues that should inform feedback. The proposed protocol makes these tasks visible as separate evaluation dimensions. In this instantiation, Gemini was closer to expert grades, while GPT-5.5 recovered more expert-reference issues with noisier and more severe assessments. The relevant decision is therefore not which model wins, but which human-in-the-loop role is appropriate for the intended assessment work.

Because both models used the same assignment materials, rubric, prompt structure, output schema, temperature setting, and nominally high reasoning setting, these differences show why the model should be treated as one component of the assessment instrument. Holding the protocol constant did not make assessment behavior

[7] https://github.com/elipcs/grading-is-easy-feedback-is-hard/blob/main/docs/tables/model_benchmark_by_category.csv

Table 2: Run-to-run stability under a fixed protocol ($n = 84$ submissions per model).

Model	ICC(2,1)	Run-pair MAE	Exact match	Within ±1.0
Gemini 3.1 Pro	0.89	0.33	2.4%	86.9%
GPT-5.5	0.90	0.39	0.0%	89.3%

Note. ICC(2,1) uses all three runs as repeated measures on the same submissions. Run-pair MAE is averaged across the three run pairs.

interchangeable: changing the model altered grading severity and diagnostic patterns.

Grade and diagnostic outcomes were descriptively associated in this study. Across runs, GPT-5.5 reported more issues than Gemini and this broader issue discovery was accompanied by systematic undergrading. High recall can support instructor review, but converting all candidate issues into deductions may lead to overly severe grading; conversely, Gemini's more selective output was associated with closer grade concordance while leaving more expert-reference issues undetected.

5.2 Why Feedback Remains Hard

RQ2 helps explain why diagnostic alignment is harder than grade concordance. A final score aggregates errors, issues in different severities, and compensating strengths, so it may remain plausible despite missing, redundant, or spurious diagnoses. Feedback is granular: it must identify the right issue, link it to code evidence, explain its relevance, and avoid irrelevant repairs.

This distinction matters because diagnostic errors are more visible in feedback than in grades. A model can assign a plausible score while omitting the specific misconception that the instructor wants the student to notice. Conversely, a model can surface many technically defensible observations and still produce poor assessment support if those observations are outside the assignment scope. This explains the complementary risks observed in the results: GPT-5.5 recovered nearly all expert-reference issues; however, its high volume of unmatched reports would likely require substantial instructor filtering; Gemini produced fewer unmatched reports; however, its missed issues would leave important feedback opportunities invisible.

Taken together, these findings reinforce an important principle from educational assessment: numerical agreement and assessment quality represent different dimensions of evaluation. While agreement measures consistency with a reference assessor, educational usefulness also depends on the quality, relevance, and pedagogical value of the generated feedback. Therefore, grade concordance should be interpreted together with diagnostic evidence rather than as a standalone indicator of assessment quality [4, 26].

Feedback is also contextual in a way that generic code review is not. In the OOP assignment studied here, some observations that would be reasonable in a professional review were not aligned with what the introductory rubric was designed to evaluate. Comments about richer Javadoc descriptions, naming refinements inherited from the starter code, or broader refactoring preferences may be technically plausible, but they can distract from the intended learning objectives when the course is assessing basic abstraction, responsibility distribution, object references, and behavioral requirements. Thus, the problem is not merely whether a reported issue is true; it is whether the issue is instructionally relevant for the submission, criterion, and course moment.

Submission S01 illustrates both sides of this problem. The expert deducted 2.0 points because adding to a List from an empty position should report "POSIÇÃO INVÁLIDA"; GPT-5.5 detected the behavior but bundled it with related issues in a 1.0-point deduction, while Gemini omitted it. Yet both models also penalized readability—imprecise Javadoc and literal numbers for GPT-5.5, and documentation clarity for Gemini—where the expert reported no readability issue. The same assessment can therefore miss or dilute a rubric-relevant failure while elevating professional-review preferences beyond the intended scope.

Submission S02 shows the complementary risk. The expert deducted 3.5 points for an off-by-one error caused by failing to subtract one from a user position before array access. Gemini identified this concrete indexing error, whereas GPT-5.5 missed it despite its higher aggregate recall. These cases reinforce that aggregate profiles describe tendencies rather than guarantees for individual submissions and that instructor review remains necessary.

The shared protocol explicitly distinguished penalizable failures from optional observations and constrained feedback to high-confidence, rubric-supported deductions. Nevertheless, GPT-5.5 frequently reported unmatched observations as penalizable deductions, particularly in reference usage, readability/documentation, responsibility division, and output format. This pattern suggests that, in this instantiation, the course-level wording did not fully prevent professional code-review behavior.

There is also a selection problem. Student-facing feedback cannot simply be the union of all detected issues, especially for novice programmers. Useful feedback requires prioritizing a small set of high-value observations, consolidating repeated symptoms under a root cause, and deciding which issues should affect the grade versus which should remain optional guidance. The two model profiles suggest different failure modes for this selection step. A high-recall model may overwhelm the reviewer with candidate issues and encourage excessive deductions, whereas a more selective model may produce a cleaner report but miss concepts that the instructor considers central.

Finally, diagnostic alignment is only the first layer of feedback quality. The RQ2 analysis evaluates whether model-reported problems correspond to expert-identified problems; it does not measure whether the final report is understandable, appropriately sequenced, actionable, or motivating for students. However, the diagnostic layer constrains all subsequent layers. If the model identifies the wrong issue, frames a peripheral issue as central, or misses a key misconception, even well-written prose may fail pedagogically. This is

why feedback remains hard even when grade concordance appears acceptable: effective feedback requires calibrated diagnosis, pedagogical prioritization, and careful translation into student-facing guidance.

The category-level patterns therefore motivate evaluating LLM support separately across issue families and retaining instructor review before model output is used as student-facing feedback. This aligns with prior work on feedback quality and LLM-based code analysis [10, 11, 13, 20, 23, 28]. In practical terms, the model output is better treated as a set of candidate diagnostic signals than as a finished feedback report.

6 Implications

The results suggest that LLM-assisted assessment should not be treated as a single-task problem. Under the evaluated protocol, grade calibration and diagnostic coverage favored different assessment behaviors; therefore, evaluation should measure both grade concordance and diagnostic alignment rather than relying on a single aggregate metric.

Within the studied setting, this supports using LLM outputs as human-in-the-loop aids rather than autonomous grades. Choosing a model is secondary to defining an AI-assisted assessment workflow: what the model output is allowed to do, who reviews it, and whether it informs grades, feedback, or both. A reusable evaluation protocol should therefore ask separate questions about grade concordance, diagnostic coverage, unmatched-report burden, and pedagogical scope. More calibrated outputs may support grade auditing, while higher-recall outputs may help instructors surface candidate issues; in both cases, outputs should remain instructor-facing before becoming student-facing feedback unless validated in the target context. A practical workflow would use model output to flag grading outliers or generate candidate feedback items, followed by instructor review to decide which observations are pedagogically relevant and how they should affect the final score. Future studies should evaluate grade concordance and diagnostic alignment separately, treating LLMs as assessment instruments that require calibration, auditing, and alignment with the educational constructs being evaluated [1, 19, 29, 31].

6.1 Threats to Validity

Regarding internal validity, operational TA assessments reflect authentic grading practice but introduce evaluator variability: each submission was graded by a single TA, and no independent TA regrading was performed. The TA–expert comparison provides an operational agreement baseline, but it is not an inter-rater reliability measure among TAs. The expert reference provides a consistent analytical benchmark, but it was produced by a single evaluator and should not be interpreted as consensus ground truth. The three technical replications per model partially characterize stochastic output variability under a fixed protocol, but they are insufficient to estimate the full output distribution with high precision; residual run-to-run instability may therefore affect both grade concordance and diagnostic alignment estimates.

Regarding construct validity, the study evaluates assessment alignment rather than learning outcomes. Grade concordance measures proximity to expert scores, while diagnostic alignment measures overlap with expert annotations. Thus, the feedback-related construct measured in RQ2 is diagnostic alignment: whether model-reported issues correspond to expert-identified problems. It is not a direct measure of the pedagogical quality of the generated report, such as tone, sequencing, actionability, student comprehension, trust, or actual use of the feedback. The semantic matching procedure also introduces measurement risk because issue equivalence involves interpretive judgment. Moreover, semantic matching was adjudicated by Claude Sonnet 5 (`claude-sonnet-5`), which is independent of both evaluated provider families (Google and OpenAI). Although the protocol used a structured taxonomy, fixed adjudication prompt, sensitivity analysis, and released matching artifacts for auditability, independent human validation of the matching decisions remains future work.

Regarding external validity, the study analyzes one Java OOP assignment in one course context and under one prompting protocol. The assignment was selected because it foregrounds design decisions, but results may differ for assignments focused primarily on algorithms, data structures, or functional correctness. The two-semester sample provides some temporal variation, but all submissions come from the same course design. The model comparison is also bounded to two flagship systems from Google and OpenAI that were the most advanced generally available offerings at the time of data collection; results may not generalize to smaller, open-weight, or subsequently released models. Reproducibility is also constrained by the use of commercial LLMs, whose provider-side implementations may evolve over time.

7 Conclusion

This paper evaluated a two-dimensional protocol for LLM-assisted, rubric-based assessment of one design-oriented OOP assignment, distinguishing grade concordance from diagnostic alignment. In this course context and prompting protocol, these dimensions diverged: Gemini more closely approximated expert-reference grades while missing relevant issues, whereas GPT-5.5 recovered more expert-reference issues while producing more reports unmatched to the expert reference and systematically lower grades. These findings do not establish general properties of LLMs across assessment settings; rather, they show why LLM-based assessment should be evaluated through separate measures of grade concordance and diagnostic alignment, and validated in context. Future work should apply the framework to other assignments, programming contexts, multi-instructor references, and student-facing feedback outcomes.

Artifact Availability

A replication package is available at https://doi.org/10.5281/zenodo.21385431 and at https://github.com/elipcs/grading-is-easy-feedback-is-hard, including the rubric, prompts, anonymized submissions, runs, expert annotations, scripts, and results.

Acknowledgements

This work was supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil.

References

[1] Daniele Agostini and Federica Picasso. 2024. Large language models for sustainable assessment and feedback in higher education: Towards a pedagogical and technological framework. *Intelligenza Artificiale* 18, 1 (2024), 121–138.

[2] Kirsti M Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15, 2 (2005), 83–102. doi:10.1080/08993400500150747

[3] Nadia Alshahwan, Jubin Chheda, Anastasia Finogenova, Beliz Gokkaya, Mark Harman, Inna Harper, Alexandru Marginean, Shubho Sengupta, and Eddy Wang. 2024. Automated Unit Test Improvement using Large Language Models at Meta. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering* (Porto de Galinhas, Brazil) *(FSE 2024)*. Association for Computing Machinery, New York, NY, USA, 185–196. doi:10.1145/3663529.3663839

[4] Susan M. Brookhart. 2013. *How to create and use rubrics for formative assessment and grading*. ASCD, Alexandria, VA.

[5] Amy Cook, Vinhthuy Phan, and Alistair Windsor. 2022. Improving TA Feedback on In-Class Coding Assignments for Introductory Computer Science. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1* (Dublin, Ireland) *(ITiCSE '22)*. Association for Computing Machinery, New York, NY, USA, 421–427. doi:10.1145/3502718.3524746

[6] Wei Dai, Jionghao Lin, Hua Jin, Tongguang Li, Yi Shan Tsai, Dragan Gasevic, and Guanliang Chen. 2023. Can Large Language Models Provide Feedback to Students? A Case Study on ChatGPT. In *Proceedings of the 2023 IEEE International Conference on Advanced Learning Technologies (ICALT)*. IEEE, 323–325. doi:10.1109/ICALT58122.2023.00100

[7] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1136–1142. doi:10.1145/3545945.3569823

[8] Oluwole Fagbohun, Nwaamaka Pearl Iduwe, Mustapha Abdullahi, Adeseye Ifaturoti, and Obinna Nwanna. 2024. Beyond Traditional Assessment: Exploring the Impact of Large Language Models on Grading Practices. *Journal of Artificial Intelligence, Machine Learning and Data Science* (2024). https://api.semanticscholar.org/CorpusID:267527222

[9] Milton Friedman. 1937. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *J. Amer. Statist. Assoc.* 32, 200 (1937), 675–701. arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1937.10503522 doi:10.1080/01621459.1937.10503522

[10] A. Gomes, D. Sousa, P. Maia, and M. Paixao. 2025. Attentionsmelling: Using Large Language Models to Identify Code Smells. In *Proceedings of the XXXIX Brazilian Symposium on Software Engineering (SBES)*. Sociedade Brasileira de Computação, Porto Alegre, RS, Brazil, 271–281. doi:10.5753/sbes.2025.9921

[11] João Gonçalves and Marcelo Maia. 2025. An Empirical Study on the Effectiveness of Iterative LLM-Based Improvements for Static Analysis Issues. In *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 382–392. doi:10.5753/sbes.2025.9964

[12] Md. Asif Haider, Ayesha Binte Mostofa, Sk. Sabit Bin Mosaddek, Anindya Iqbal, and Toufique Ahmed. 2024. Prompting and Fine-tuning Large Language Models for Automated Code Review Comment Generation. arXiv:2411.10129 [cs.SE] https://arxiv.org/abs/2411.10129

[13] John Hattie and Helen Timperley. 2007. The Power of Feedback. *Review of Educational Research* 77, 1 (2007), 81–112. arXiv:https://doi.org/10.3102/003465430298487 doi:10.3102/003465430298487

[14] Sture Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics* 6, 2 (1979), 65–70. http://www.jstor.org/stable/4615733

[15] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '10)*. Association for Computing Machinery, New York, NY, USA, 86–93. doi:10.1145/1930464.1930480

[16] Michael T. Kane. 1992. An argument-based approach to validity. *Psychological Bulletin* 112, 3 (1992), 527–535.

[17] Enkelejda Kasneci, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stephan Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, and Gjergji Kasneci. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences* 103 (2023), 102274. doi:10.1016/j.lindif.2023.102274

[18] M. G. Kendall and B. Babington Smith. 1939. The Problem of m Rankings. *The Annals of Mathematical Statistics* 10, 3 (1939), 275–287. http://www.jstor.org/stable/2235668

[19] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Trans. Comput. Educ.* 19, 1, Article 3 (Sept. 2018), 43 pages. doi:10.1145/3231711

[20] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. 2023. Exploring the Potential of Large Language Models to Generate Formative Programming Feedback . In *2023 IEEE Frontiers in Education Conference (FIE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–5. doi:10.1109/FIE58773.2023.10343457

[21] Smitha S. Kumar, Michael A. Lones, Manuel Maarek, and Hind Zantout. 2025. Navigating the Landscape of Automated Feedback Generation Techniques for Programming Exercises. *ACM Trans. Comput. Educ.* 25, 4, Article 54 (Oct. 2025), 29 pages. doi:10.1145/3764593

[22] Michael Kölling, Bruce Quig, Andrew Patterson, and John Rosenberg. 2003. The BlueJ System and its Pedagogy. *Computer Science Education* 13, 4 (2003), 249–268. arXiv:https://doi.org/10.1076/csed.13.4.249.17496 doi:10.1076/csed.13.4.249.17496

[23] Keila Lucas, Rohit Gheyi, Elvys Soares, Márcio Ribeiro, and Ivan Machado. 2024. Evaluating Large Language Models in Detecting Test Smells. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 672–678. doi:10.5753/sbes.2024.3642

[24] Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaojing Shi. 2024. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Trans. Comput. Educ.* 24, 1, Article 10 (Feb. 2024), 43 pages. doi:10.1145/3636515

[25] Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaojing Shi. 2025. How Consistent Are Humans When Grading Programming Assignments? *ACM Trans. Comput. Educ.* 25, 4, Article 49 (Sept. 2025), 37 pages. doi:10.1145/3759256

[26] Samuel Messick. 1989. Validity. In *Educational Measurement* (3rd ed.), Robert L. Linn (Ed.). Macmillan, New York, 13–103.

[27] Gustavo Pinto, Isadora Cardoso-Pereira, Danilo Monteiro, Danilo Lucena, Alberto Souza, and Kiev Gama. 2023. Large Language Models for Education: Grading Open-Ended Questions Using ChatGPT. In *Anais do XXXVII Simpósio Brasileiro de Engenharia de Software* (Campo Grande/MS). SBC, Porto Alegre, RS, Brasil, 293–302. https://sol.sbc.org.br/index.php/sbes/article/view/28298

[28] Igor Regis da Silva Simões and Elaine Venson. 2025. Measuring how changes in code readability attributes affect code quality evaluation by Large Language Models. In *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software (SBES 2025) (SBES 2025)*. Sociedade Brasileira de Computação, 13–24. doi:10.5753/sbes.2025.9603

[29] Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information Processing & Management* 45, 4 (2009), 427–437. doi:10.1016/j.ipm.2009.03.002

[30] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. 2016. Designing a rubric for feedback on code quality in programming courses. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '16)*. Association for Computing Machinery, New York, NY, USA, 160–164. doi:10.1145/2999541.2999555

[31] Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S. Yu, and Qingsong Wen. 2025. Large Language Models for Education: A survey and outlook. *IEEE Signal Processing Magazine* 42, 6 (2025), 51–63. doi:10.1109/MSP.2025.3594309

[32] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968

[33] Bingxu Xiao, Yunwei Dong, Yiqi Tang, Manqing Zhang, Yifan Zhou, Chunyan Ma, and Yepang Liu. 2026. OODEval: Evaluating Large Language Models on Object-Oriented Design. *arXiv preprint arXiv:2601.07602* (2026). doi:10.48550/arXiv.2601.07602

[34] Zhiqiang Yuan, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, Xin Peng, and Yiling Lou. 2024. Evaluating and Improving ChatGPT for Unit Test Generation. *Proc. ACM Softw. Eng.* 1, FSE, Article 76, 24 pages. doi:10.1145/3660783

[35] Humberto Augusto Piovesana Zanetti, Marcos Augusto Francisco Borges, and Ivan Luiz Marques Ricarte. 2023. ComFAPOO: Método de Ensino de Programação Orientada à Objetos Baseado em Aprendizagem Significativa e Computação Física. *Revista Brasileira de Informática na Educação* 31 (jan. 2023), 01–30. doi:10.5753/rbie.2023.2851