Co-TA: Streamlining Automated Grading with Generative Rubrics and Distribution Simulation

Helen Jin HELENJIN@SEAS.UPENN.EDU and **Albert Chen** ALBERT.CHEN@SEAS.UPENN.EDU
Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, United States.

Abstract

Large Language Models (LLMs) offer great potential to reduce Teaching Assistant workloads through automated grading. However, their rapid integration shifts the primary challenge from technical scalability to responsible governance, as LLMs frequently diverge from expert human graders on complex rubrics. Existing Human-in-the-Loop (HITL) systems focus on micro-level calibration for individual submissions, leaving instructors blind to how rubrics alter class-wide grade distributions. To address this gap, we introduce **Co-TA**, a deployable web application that streamlines the grading workflow while shifting the paradigm to rubric co-design and fairness auditing. Featuring batch uploads and uncertainty flagging, Co-TA allows instructors to generate rubric variations and immediately simulate their impact against synthetic student personas. By visualizing resulting grade distributions prior to live grading, Co-TA serves as a critical fairness auditing mechanism, ensuring AI assessment remains transparent, verifiable, and firmly under human pedagogical control.[1]

Keywords: Automated Grading, Large Language Models, Generative Rubrics, Fairness Auditing, AI in Education

1. Introduction

To alleviate the massive manual grading burden placed on teaching staff in large-enrollment courses, instructors are increasingly turning to Large Language Models (LLMs) to automate summative assessments. However, while LLMs offer unprecedented scalability for grading complex open-ended assignments, they introduce profound risks regarding fairness, strictness drift, and pedagogical alignment. To safely lower this barrier, educators need an easy way to create rubrics based on specific instructor specifications, and the ability to test out different rubrics to see the resulting grade distributions before finalizing grades.

Two recent findings sharpen this question. First, Mahinpei et al. (2026) show that when LLMs are asked to apply a grading rubric to complex, open-ended student work, there is substantial disagreement with human TAs. This is not because the model lacks knowledge, but because rubric *interpretation* is inherently ambiguous. Second, Liu et al. (2025) demonstrate that hybrid "Human TA + Expert LLM" configurations consistently outperform either humans or AI alone. These interpretation issues extend beyond formal proofs and code. Recent studies reveal that LLMs exhibit systemic leniency when awarding partial credit for short answers (Deng et al., 2025), struggle with scoring range compression in essays (Mathew et al., 2026), and heavily penalize non-native writing styles (Jadhav et al., 2026).

These findings motivate a shift in the design of human-in-the-loop (HITL) systems. The current state of the art focuses on *micro-level calibration*: aligning AI-generated grades with instructor expectations on a per-submission basis. Systems such as Avalon (Armfield et al., 2025) route low-confidence items to human reviewers, while CHiL(L)Grader (Raikote et al., 2026) introduces iterative dialogue to refine individual assessments (Chu et al., 2025). While valuable, these approaches address fairness *after* grades have been produced—they calibrate the

1. *Note:* This paper is submitted as a work-in-progress. The current evaluation is mostly illustrative and uses synthetic student personas; future work will evaluate Co-TA with real student submissions, instructor feedback, historical grade comparisons, and real classroom deployment.

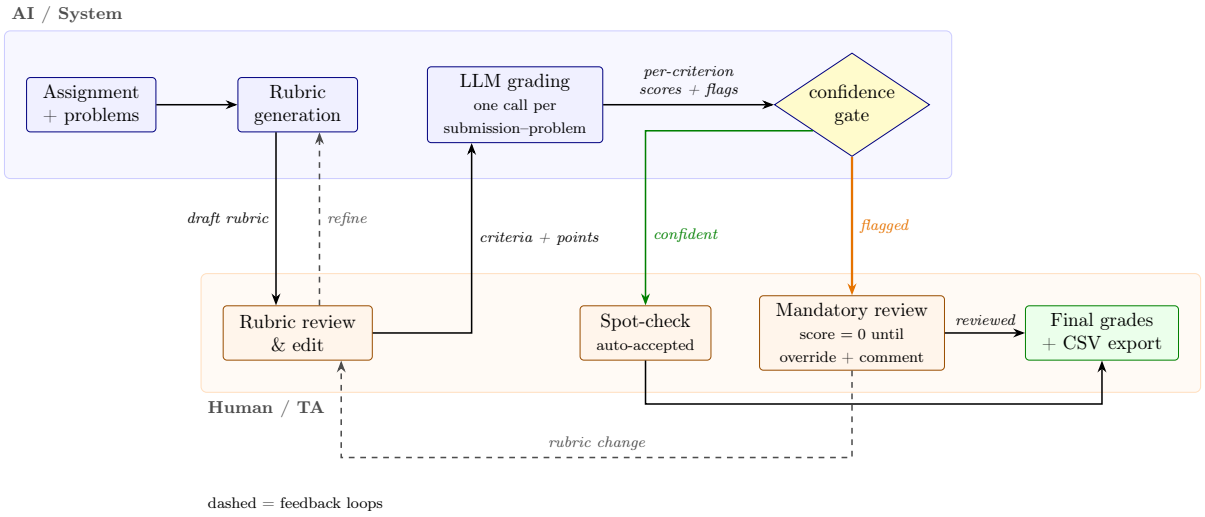

Figure 1: The Co-TA pipeline, which includes rubric generation, review, automated grading, and human validation.

outputs of a fixed rubric rather than questioning the rubric itself. We argue that to truly reduce TA workload safely, a complementary, *macro-level* perspective is needed. Before any student work is graded en masse, instructors should be able to ask: "If I adopt this rubric, what will the grade distribution look like across my class?"

In this paper we present **Co-TA** (Co-Teaching Assistant), a web-based tool suite that operationalises this idea. While Co-TA's primary motivation is streamlining the grading workflow for educators, it supports a unique and natural three-stage workflow to do so safely: (1) *Generative Rubric Design*, drafting rubric variations from an instructor's natural-language specification; (2) *Distribution Simulation*, applying candidate rubrics to a bank of synthetic student submissions and visualising the resulting distributions; and (3) *Instructor Auditing*, adjusting rubric criteria iteratively. Our contributions are:

- We introduce an end-to-end grading suite designed to reduce TA workload while incorporating *grade distribution simulation* as a novel safeguard.
- We describe the architecture and workflow of Co-TA, a deployable tool for rubric co-design. Co-TA is available at [https://github.com/albert239825/Co-TA].
- We present a demonstrative case-study evaluation using five synthetic student personas that probe distinct fairness dimensions. We plan to deploy Co-TA in a real classroom setting at the University of Pennsylvania in an upcoming semester.

2. The Co-TA System Architecture

Recent architectural advancements in AI grading, such as StepGrade (Akyash et al., 2025) and the "Grade Like a Human" framework (Xie et al., 2024), have focused heavily on maximizing the micro-level grading accuracy of LLMs. These frameworks construct sophisticated Chain-of-Thought (CoT) and dynamic prompt pipelines to ensure that an AI's grade for a single paper mirrors a human expert's grade as closely as possible.

In contrast, Co-TA is designed as a macroscopic governance layer (see Figure 1). Co-TA does not claim a novel algorithmic prompt to perfectly grade a paper; rather, it provides a dashboard that visualizes the population-level consequences of a grading policy prior to live deployment. To achieve this, Co-TA is built on a modern, deployable web stack consisting of

Next.js 14 (App Router), SQLite, and Drizzle ORM. This lightweight architecture allows the application to be deployed rapidly within institutional environments while maintaining strict local data persistence for student records.

The Co-TA workflow begins with the instructor inputting their assignment specifications and a gold-standard solution. The system's Generative Rubric Engine drafts multiple rubric variations. Once a rubric is simulated and approved, the TA can upload student submissions in a batch (e.g., PDFs or code files) for high-throughput, schema-enforced automated grading, drastically reducing manual assessment time. Co-TA enforces a fixed output schema that separates `total_score`, `points_awarded`, and `feedback` for every rubric criterion, allowing the frontend to instantly aggregate the numerical outputs and chart the grade distributions. We treat this as an engineering convenience rather than a contribution: schema validation improves parseability and aggregation, but does not by itself guarantee that a grade is correct or that the model has faithfully applied the rubric.

Human-in-the-Loop Flagging Co-TA does not finalize every grade autonomously. The grading prompt instructs the model to flag a submission for human review, rather than commit a score, whenever a condition falls outside the rubric's intended scope. The same schema carries a `flagged_for_review` field and a short rationale, so flagged items surface directly in the dashboard for a TA to resolve. The flag fires uniformly across all submissions on: (i) low model confidence or genuine ambiguity in how a criterion applies; (ii) a correct but unconventional solution the rubric does not anticipate; and (iii) stylistic features the rubric is not meant to assess, such as non-standard or non-native English that could confound scoring of the underlying work. Co-TA does not infer student demographics or algorithmically adjust grades for linguistic factors; it surfaces ambiguous cases and leaves any equity decision to the instructor.

3. Case Study: Simulating Fairness in CS1

While Co-TA is designed as a general-purpose grading suite, we present the following case study as an initial demonstration of the Co-TA workflow (but with future plans for a validated classroom deployment). We use a classic introductory computer science (CS1) problem, *finding the index of the first non-repeating character in a string*, which has an expected optimal solution that runs in $\mathcal{O}(n)$ time, to illustrate how synthetic-persona analysis can help instructors inspect rubric behavior prior to use.

Consider a TA tasked with grading hundreds of CS1 submissions. Instead of manually grading them all, the TA uses Co-TA to generate two rubric variations: Rubric A (Strict) and Rubric B (Holistic). Before batch-grading real students, the TA simulates these rubrics against five synthetic personas engineered to represent dominant error clusters identified in recent learner profile studies, as shown in Table 1 (Geng et al., 2023; Hoq et al., 2025; Alzahrani and Vahid, 2021). This methodology follows the recommendations of Zhao et al. (2026), who highlighted that educators struggle to articulate precise rubrics and require systematic evaluation tools.

Persona	**Rubric A Strict**	**Rubric B Holistic**
Logic Master	0	95
Clean Coder	30	50
Brute Forcer	70	92
Off-by-One Victim	70	88
Over-Complicator	100	94

Table 1: Grades assigned by `gpt-5-mini` to five simulated personas under a strict, execution-oriented rubric (A) and a holistic, intent-aware rubric (B). Scores are out of 100.

As shown in Table 1, the strict rubric (A) is dominated by whether the code compiles and produces correct output. The Logic Master, whose algorithm is correct but whose submission contains a syntax error, fails outright (0), while the Over-Complicator passes every functional check and earns full marks (100) despite needlessly wrapping a simple task in a class. The holistic rubric (B) compresses this spread by crediting underlying intent and penalizing poor design. The Logic Master recovers almost all of its score (95) once the grader looks past the syntax error to the sound logic, the Over-Complicator is trimmed only slightly (94) for violating the KISS principle, and the partially-correct attempts (Clean Coder, Off-by-One Victim, and Brute Forcer) each gain partial credit (50, 88, and 92). By surfacing these divergent outcomes *before* real grades are assigned, Co-TA lets the instructor see how a rubric choice will distribute credit and make a data-informed policy decision.

4. Discussion

The deployment of automated grading systems introduces a profound tension in education: the drive for scalability and reduced TA workload often conflicts with the necessity of pedagogical fairness and instructor oversight. Unchecked AI grading systems frequently suffer from context loss and hallucination, directly undermining student trust and assessment validity (Nieto-Cardenas et al., 2026).

When Rubric Simulation is Useful Co-TA is most useful in settings where instructors are uncertain whether a rubric will produce the intended grading behavior before it is applied to student submissions. For example, distribution simulation can help identify assignment questions that are disproportionately difficult, rubric criteria that penalize minor implementation differences too harshly, or cases where a correct but unconventional solution receives an unexpectedly low score. The goal here is not to automatically change grades based on the simulated distribution, but to provide instructors with an opportunity to inspect the rubric before deployment.

One potential use case is pre-assessment calibration. If Co-TA reveals that a particular problem leads to systematically low simulated scores across otherwise competent student personas, instructors may decide to revise the rubric, adjust the point weighting, add clarifying instructions, or remove the problem from the assessment. Similarly, if a rubric assigns high scores to solutions with important conceptual gaps, instructors can strengthen the rubric before grading begins. In this way, Co-TA supports rubric auditing as an instructor-centered decision aid rather than an automatic grading policy.

Conclusion As the capabilities of Large Language Models continue to expand, the goal of educational technology should not simply be to replace human grading, but to augment instructor agency. Ultimately, Co-TA demonstrates that alleviating the massive manual grading burden on TAs does not have to come at the cost of pedagogical fairness. By combining easy rubric generation, batch processing, and distribution simulation, Co-TA empowers teaching staff to act as pedagogical policy-makers rather than manual graders. We further discuss limitations and future work in Appendix A.

References

Mohammad Akyash, Kimia Zamiri Azar, and Hadi Mardani Kamali. Stepgrade: Grading programming assignments with context-aware llms. *arXiv preprint arXiv:2503.20851*, 2025.

Nabeel Alzahrani and Frank Vahid. Common logic errors for programming learners: A three-decade literature survey. *ASEE Annual Conference and Exposition*, 2021.

Derek Armfield, Eason Chen, Asilbek Omonkulov, Xinyi Tang, Jionghao Lin, Erik Thiessen, and Kenneth Koedinger. Avalon: A human-in-the-loop llm grading system with instructor

calibration and student self-assessment. In *International Conference on Artificial Intelligence in Education (AIED)*. Springer, 2025.

Yucheng Chu, Hang Li, Kaiqi Yang, Yasemin Copur-Gencturk, and Jiliang Tang. Llm-based automated grading with human-in-the-loop. *arXiv preprint arXiv:2504.05239*, 2025.

Haotian Deng, Chris Farber, Jiyoon Lee, and David Tang. Rubric-conditioned llm grading: Alignment, uncertainty, and robustness. *arXiv preprint arXiv:2601.08843*, 2025.

Chuqin Geng, Wenwen Xu, Yingjie Xu, Brigitte Pientka, and Xujie Si. Identifying different student clusters in functional programming assignments: From quick learners to struggling students. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2023.

Muntasir Hoq, Ananya Rao, Reisha Jaishankar, Krish Piryani, Nithya Janapati, Jessica Vandenberg, Bradford Mott, Narges Norouzi, James Lester, and Bita Akram. Automated identification of logical errors in programs: Advancing scalable analysis of student misconceptions. *arXiv preprint arXiv:2505.10913*, 2025.

Rudra Jadhav, Janhavi Danve, and Sonalika Shaw. Implicit grading bias in large language models: How writing style affects automated assessment across math, programming, and essay tasks. *arXiv preprint arXiv:2603.18765*, 2026.

Ziqi Liu, Bolan Yang, and Shizhen Huang. Exploring the impact of different assistance approaches on students' performance in engineering lab courses. *Education Sciences*, 15:1443, 10 2025. doi: 10.3390/educsci15111443.

Romina Mahinpei, Sofiia Druchyna, and Manoel Horta Ribeiro. When llms help – and hurt – teaching assistants in proof-based courses. *arXiv preprint arXiv:2602.23635*, 2026.

Jerin George Mathew, Sumayya Taher, Anindita Kundu, and Denilson Barbosa. Llms do not grade essays like humans. *arXiv preprint arXiv:2603.23714*, 2026.

Juliana Nieto-Cardenas, Erin Joy Kramer, Peter Kurto, Ethan Dickey, and Andres Bejarano. Owlgorithm: Supporting self-regulated learning in competitive programming through llm-driven reflection. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2026.

OpenAI. GPT-5 system card, 2025. URL https://arxiv.org/abs/2601.03267.

Pranav Raikote, Korbinian Randl, Ioanna Miliou, Athanasios Lakes, and Panagiotis Papapetrou. Chil(l)grader: Calibrated human-in-the-loop short-answer grading. *arXiv preprint arXiv:2603.11957*, 2026.

Wenjing Xie, Juxin Niu, Chun Jason Xue, and Nan Guan. Grade like a human: Rethinking automated assessment with large language models. *arXiv preprint arXiv:2405.19694*, 2024.

Victor Zhao, Max Fowler, Yael Gertner, Seth Poulsen, Matthew West, and Mariana Silva. Ai-supported grading and rubric refinement for free response questions. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2026.

Appendix A. Limitations and Future Work

Limitations While our CS1 case study demonstrates the utility of simulation via synthetic personas, we note that the current iteration of the system relies entirely on these engineered profiles to project class distributions. A primary limitation is that instructors must manually ensure these synthetic personas accurately represent the actual distribution of misconceptions within their specific cohort, which may vary widely across different universities or assignment types (e.g. short essays vs. coding tasks). We are currently developing a feature to help instructors more easily generate, refine, and simulate personas for their own course contexts.

Also, our current personas primarily capture programming-related failure modes rather than demographic, linguistic, accessibility-related, or educational-background differences. We do not yet know how salient these dimensions are for the specific rubric-simulation setting studied here, or how best to operationalize them without relying on reductive or stereotyped personas. As a result, this case study should be interpreted as an initial demonstration of Co-TA's simulation workflow rather than a comprehensive fairness evaluation. Future work should examine whether and how these additional dimensions affect rubric behavior, ideally using instructor input and real classroom data.

Moreover, although our original motivation was primarily for computer science courses, we foresee Co-TA being especially useful in other STEM domains, where instructors often evaluate structured reasoning, procedural correctness, and partial-credit solutions. We do note however that extensions may require domain-specific personas and further instructor validation to ensure that the simulations capture meaningful mistakes in each context.

Future Work We see future iterations of Co-TA extending the tool suite beyond rubric simulation by integrating historical data analysis. We plan to develop pipelines that automatically generate representative synthetic personas by clustering actual historical student submissions from previous semesters across various non-programming domains. Furthermore, future work will explore the generation and comparative analysis of a wider variety of rubric structures (e.g., single-point versus highly analytic rubrics) to evaluate their downstream impacts on AI grading reliability.

It is also paramount to evaluate Co-TA across both pre-deployment review settings and real-world classroom deployments. We intend to deploy Co-TA in a real classroom setting at the University of Pennsylvania in an upcoming semester. As an initial step, we will conduct instructor walk-throughs and expert rubric reviews in which instructors inspect generated rubrics, simulated student outcomes, and uncertainty flags before classroom use. During these sessions, we will ask instructors to identify ambiguous criteria, unexpected grading behavior, and rubric choices that may require revision. We will then analyze both the revisions made to the rubrics and instructors' qualitative feedback to evaluate whether Co-TA supports pre-deployment rubric auditing.

Beyond these controlled reviews, future work should also evaluate Co-TA in live, large-enrollment environments to assess the usability of the generative rubric dashboard and uncertainty-flagging features under realistic instructional constraints. These deployments will also test how the system scales to large course settings while maintaining sufficiently low latency for practical instructor use.

Appendix B. Aligning with iRAISE Principles

Our system directly addresses the iRAISE workshop pillars of "Responsible AI Standards" and "Classroom Impact." By treating a grading rubric not as a static set of rules, but as a dynamic policy whose impact can be simulated, Co-TA functions as an active fairness auditing mechanism. Instructors are no longer passive consumers of AI-generated grades; they are active co-designers who use data-driven simulations to prevent "algorithmic shock" before grades are deployed.

Furthermore, the use of rigid output schemas (Zod) ensures that the AI's evaluations remain structurally aligned with the instructor's pedagogical intent.

Appendix C. Co-TA Interface

This appendix walks through the Co-TA interface across the core grading workflow: rubric authoring, batch upload, the pre- and post-grading dashboards, and the per-submission review view.

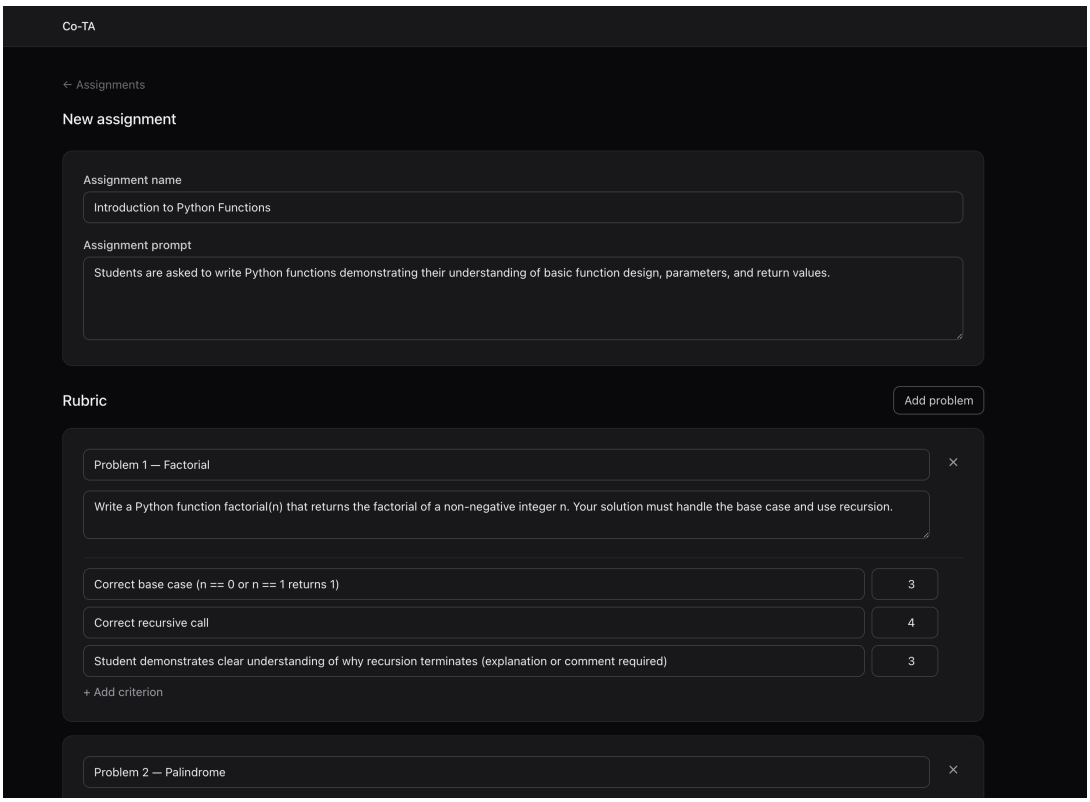

Figure 2: Assignment creation. The instructor defines the assignment name and prompt, then specifies a per-problem rubric with weighted criteria that serve as the grading contract for the LLM.

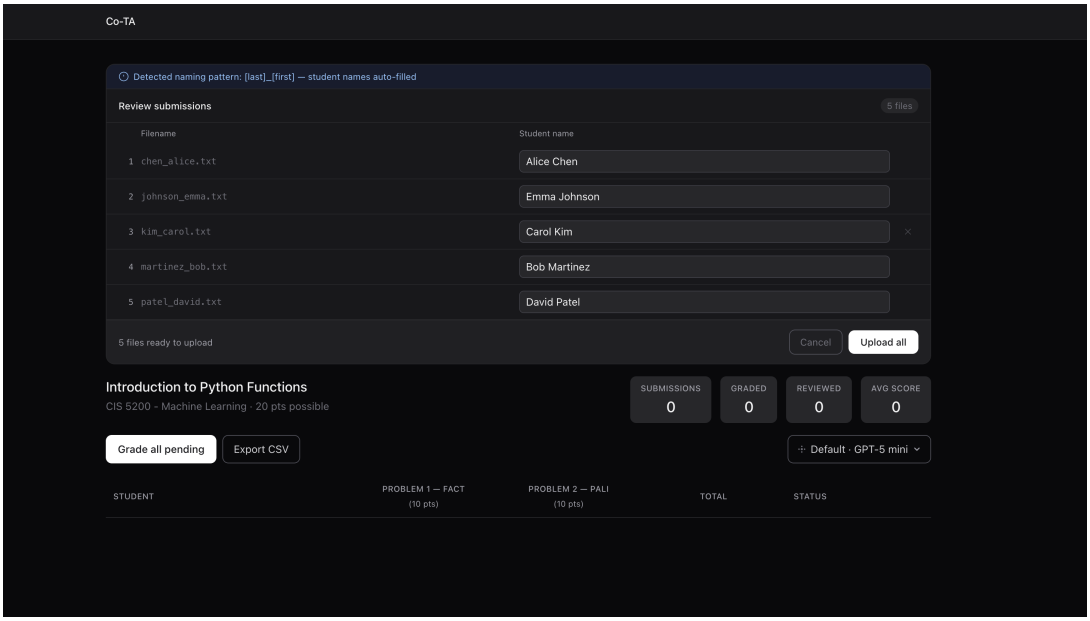

Figure 3: Batch upload. Co-TA detects the filename naming pattern (here, `[last]_[first]`) and auto-fills student names, which the instructor can review and edit before committing the upload.

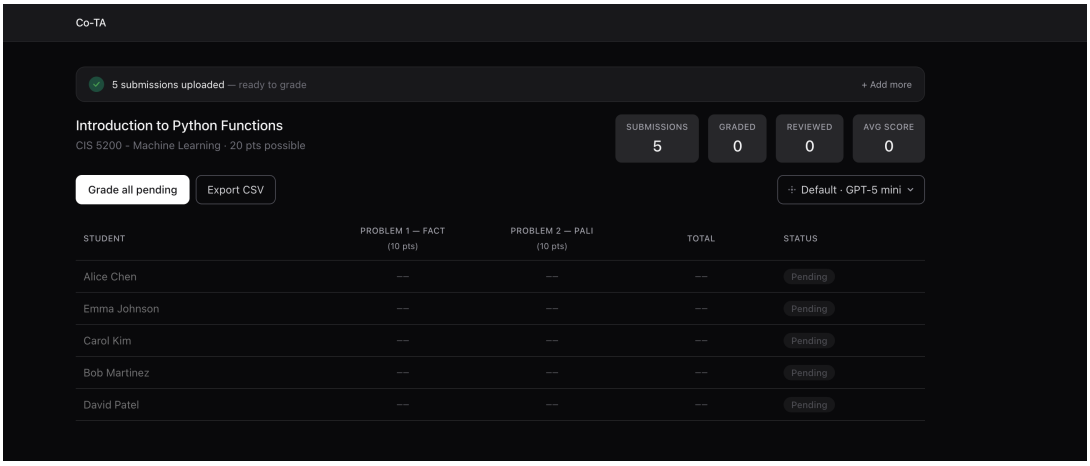

Figure 4: Pre-grading dashboard. Submissions are uploaded and marked pending; per-problem and total scores remain blank until the instructor triggers "Grade all pending."

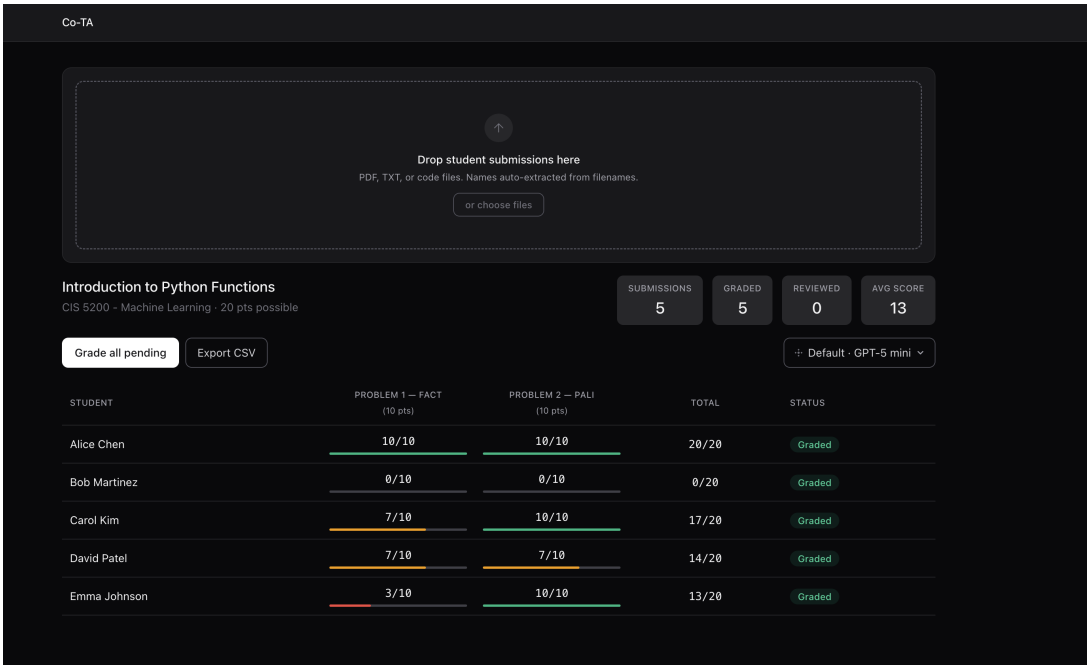

Figure 5: Post-grading dashboard. Each row shows per-problem and total scores with color-coded performance bars, while header tiles (Submissions, Graded, Reviewed, Avg Score) summarize the cohort.

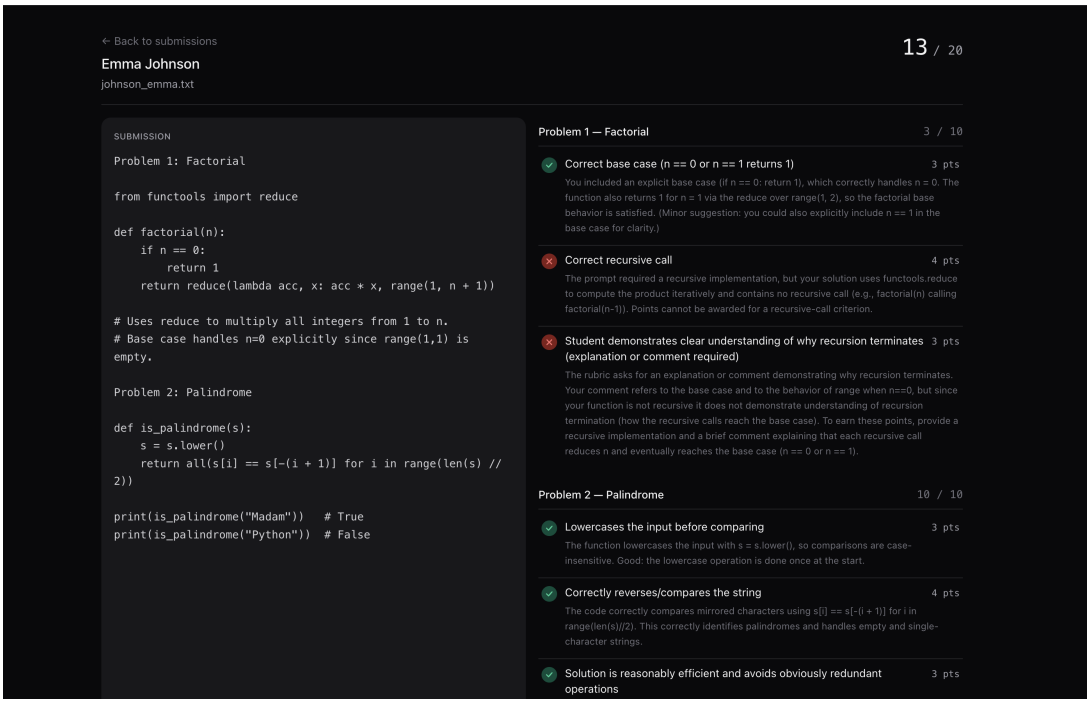

Figure 6: Per-submission review. The student's code is displayed alongside criterion-level scores and the model's natural-language justification for each award or deduction, allowing the instructor to verify reasoning before finalizing grades.

Appendix D. Pipeline

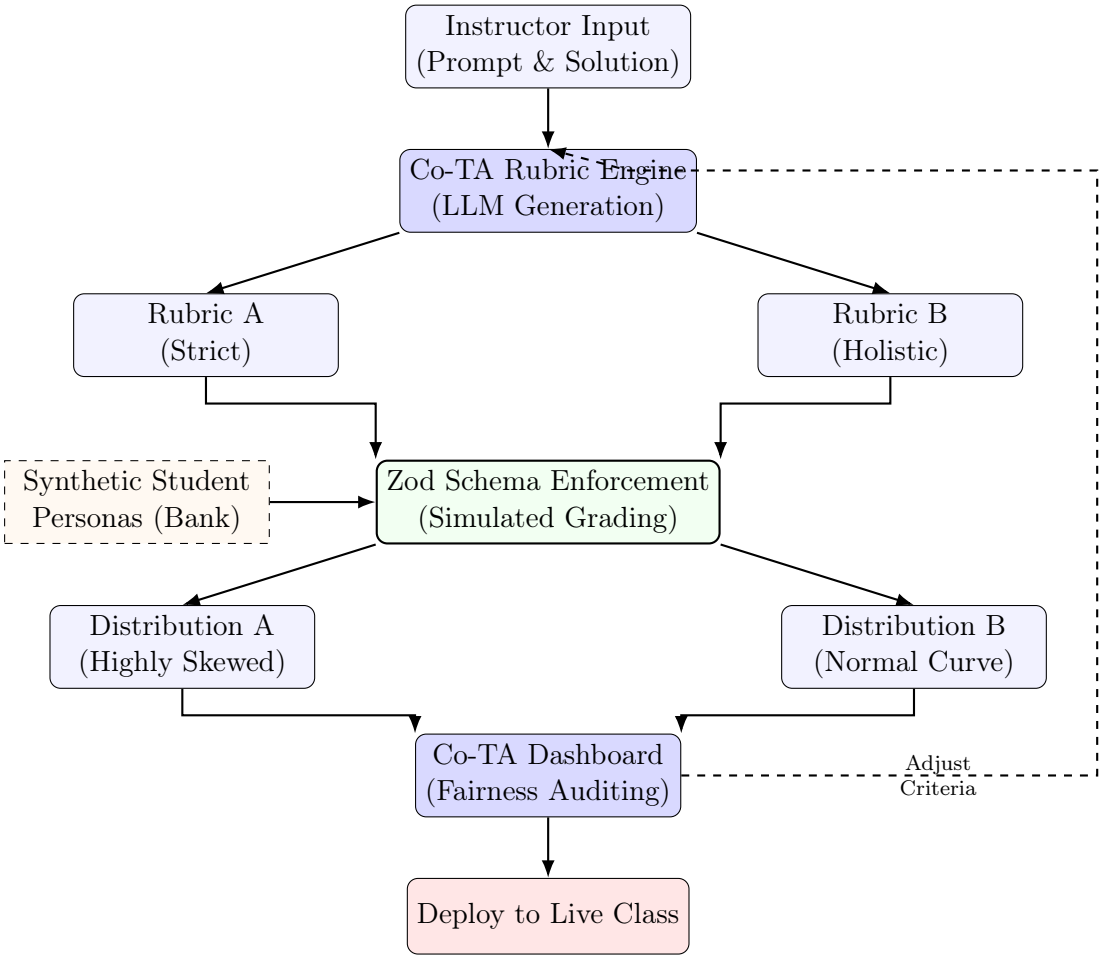

Figure 7: The Co-TA workflow combining generative rubric design, synthetic student simulation via Zod schema enforcement, and macroscopic fairness auditing.

Appendix E. Synthetic Personas and Zod Outputs

This appendix provides the concrete artifacts used during the simulation case study, demonstrating the Python code used for all five synthetic student personas, alongside the structured JSON outputs generated by the Co-TA LLM pipeline.

E.1. Persona 1: The Logic Master with Bad Syntax

This student correctly identifies the algorithmic approach ($\mathcal{O}(n)$ hash map) but contains a fatal syntax error (missing colon on line 1).

```python
# Missing the colon at the end of the function definition
def first_unique(s)
    counts = {}
    for char in s:
        counts[char] = counts.get(char, 0) + 1
    for i, char in enumerate(s):
        if counts[char] == 1:
            return i
    return -1
```

Persona 1 Code

Structured LLM Output: Rubric A (Strict)

Under the strict rubric, Co-TA's Zod-validated schema constrains the LLM to emit a binary `earned` verdict and feedback per criterion. The submission fails both criteria because of the compilation error, for a computed total of 0/100.

```
{
  "scores": [
    {
      "criterionId": "exec-functionality",
      "earned": false,
      "feedback": "SyntaxError: missing ':' after the def statement, so the code never compiles or runs against the test cases."
    },
    {
      "criterionId": "algorithmic-efficiency",
      "earned": false,
      "feedback": "Efficiency cannot be assessed because the submission does not execute."
    }
  ]
}
```

Co-TA grader output under Rubric A

Structured LLM Output: Rubric B (Holistic)

Under the holistic rubric, Co-TA isolates the execution failure from the conceptual logic: the logic and code-quality criteria are earned while only the execution criterion is lost, recovering most of the credit for a computed total of 80/100. Because each criterion is scored all-or-nothing, the missing colon forfeits the entire execution criterion rather than a few points.

```
{
  "scores": [
    {
      "criterionId": "logic-intent",
```

```
        "earned": true,
        "feedback": "Correct two-pass hash-map approach that finds the first non-repeating character in O(n)."
      },
      {
        "criterionId": "execution-syntax",
        "earned": false,
        "feedback": "Does not run: a missing ':' at the end of the def line throws a SyntaxError."
      },
      {
        "criterionId": "code-quality",
        "earned": true,
        "feedback": "Clear variable names (counts, char) and clean, consistent indentation."
      }
    ]
  }

```

Co-TA grader output under Rubric B

E.2. Persona 2: The Clean Coder with Wrong Logic

This submission features excellent documentation, type hints, and naming conventions, but fundamentally fails the logic by returning the first repeating character instead of the unique one.

```python
def first_unique_clean(s: str) -> int:
    """
    Finds the first unique character in a string.
    Returns the index if found, otherwise returns -1.
    """
    freq_map = {}
    # Flawed logic: returns first repeating instead of first unique
    for index, current_char in enumerate(s):
        if current_char in freq_map:
            return index
        freq_map[current_char] = 1
    return -1
```

Persona 2 Code

E.3. Persona 3: The Brute Forcer

This code correctly outputs the expected answer and passes unit tests, but utilizes the `.count()` method inside a loop, resulting in a suboptimal $\mathcal{O}(n^2)$ time complexity.

```python
def first_unique_brute(s):
    for i in range(len(s)):
        # .count() makes this an O(n^2) operation
        if s.count(s[i]) == 1:
            return i
    return -1
```

Persona 3 Code

E.4. Persona 4: The Off-by-One Victim

This submission implements the correct hash map algorithm but contains a minor index initialization error, starting the second loop at `1` instead of `0`, skipping the first character.

```python
def first_unique_off_by_one(s):
    counts = {}
    for char in s:
        counts[char] = counts.get(char, 0) + 1
    # Flaw: skips index 0
    for i in range(1, len(s)):
        if counts[s[i]] == 1:
            return i
    return -1
```

Persona 4 Code

E.5. Persona 5: The Over-Complicator

This code solves the problem accurately but violates standard software engineering practices (the KISS principle) by creating unnecessary classes and instance state for a simple function.

```python
from collections import Counter
class UniqueCharacterFinder:
    def __init__(self, string_data):
        self.data = string_data
        self.metrics = Counter(self.data)

    def execute_search(self):
        for idx in range(len(self.data)):
            if self.metrics[self.data[idx]] == 1:
                return idx
        return -1

def first_unique_complicated(s):
    finder = UniqueCharacterFinder(s)
    return finder.execute_search()
```

Persona 5 Code

E.6. Model and Prompting Details

All grades in Table 1 were produced by `gpt-5-mini` (OpenAI, 2025) through the Chat Completions API, using the provider's default decoding settings (temperature not overridden, no fixed seed). Each grading call uses a two-message prompt: a *system* message containing the full rubric text (Rubric A, strict; or Rubric B, holistic), and a *user* message containing only the student submission with a fixed "Grade the following code out of 100" instruction; both prompts are held constant across all personas and are reproduced below. The grader is constrained to a typed schema rather than free text, and no score is stored or hand-edited, so every reported cell is the model's output under the stated prompt. We leave repeated-run stability analysis to future work.

Prompt excerpt for synthetic grading simulation

User prompt

Grade the following code out of 100:

```
{the persona's code}
```

System prompt: Rubric A (*Strict*)

You are a strict grading assistant. Grade the student's Python code based only on the following criteria:

1. **Execution and Functionality (70 pts).** The code must compile without syntax errors and return the exact correct output for all test cases. Do not award partial credit if the code fails to compile.
2. **Algorithmic Efficiency (30 pts).** If the code compiles, evaluate whether it runs in $\mathcal{O}(n)$ time. If it does not compile, award 0 points because efficiency cannot be tested.

System prompt: Rubric B (*Holistic*)

You are an expert human Teaching Assistant. Grade the student's Python code using a step-by-step, holistic approach:

1. **Algorithmic Logic and Intent (50 pts).** Award high partial credit when the underlying algorithm is sound, even if a typo prevents execution. Award 0 points if the core logic is completely wrong.
2. **Execution and Syntax (20 pts).** Deduct points for syntax errors or off-by-one errors. A single missing colon should only cost about 5 points if the rest of the code is sound.
3. **Code Quality and Efficiency (30 pts).** Deduct points for unnecessary complexity and for using $\mathcal{O}(n^2)$ nested loops when an $\mathcal{O}(n)$ solution is expected. Do not reward pretty formatting if the underlying logic is broken.

Appendix F. Governance and Deployment Risks

Co-TA is intended to support instructors in auditing rubrics before deployment, not to replace instructor judgment or autonomously assign final grades. Classroom use raises governance concerns around student consent, educational data privacy, data retention, and accountability for grading decisions. Because these requirements vary across institutions, Co-TA should be deployed under local data-governance policies specifying what data may be shared, retained, accessed, or processed by third-party model providers. Co-TA should also maintain audit logs of rubric revisions, simulated grading outputs, uncertainty flags, instructor overrides, and any grading-related changes. These safeguards are important because even instructor-facing systems can shape assessment decisions. Accordingly, Co-TA should be used only as a decision-support tool, with final authority and responsibility remaining with human instructors and TAs.