Exploring the Potential of Locally Run Large Language (AI) Models for Automated Grading in Introductory Computer Science Courses

Samuel B Mazzone
Computer Science
Marquette University
Milwaukee, Wisconsin
samuel.mazzone@marquette.edu

Jack Forden
Computer Science
Marquette University
Milwaukee, Wisconsin
jack.forden@marquette.edu

Dennis Brylow
Computer Science
Marquette University
Milwaukee, Wisconsin
dennis.brylow@marquette.edu

***Abstract*—This innovative practice full paper describes the effectiveness of self-hosted large language models (LLMs) in assisting with the automatic grading of CS1 assignments.**

Educators often rely on automated review of student code submissions in larger courses. Despite recent advancements, current systems primarily focus on assessing functionality, with important aspects such as code structure, efficiency, and style often relegated to secondary foci. LLMs provide an increasingly attractive addition to these systems to enhance those overlooked areas. Prior research has shown LLM's capable of assisting students in understanding and resolving programmer error messages, correcting syntax errors, providing enhanced explanations of code segments, or even generating code.

The absence of freely available, purpose-designed LLMs for grading and providing feedback on code submissions prevents widespread adoption by educators.

Remotely-hosted systems, such as fine-tuned GPT models, have shown promise, yet the associated risks of privacy breaches, ethical considerations, and recurring costs make this approach unfeasible as a universal solution. To mitigate these concerns, self-hosted open-source models are an alternative that can operate on consumer-grade hardware and prevent some privacy and security concerns. While no purpose-built solution yet exists, it is unclear if any existing models are powerful enough to facilitate automated grading. To explore these questions, we present a two-phase analysis, leveraging real grading data from a semester length, introductory CS1 course with 124 students and nine programming projects. Nine stable LLM models were selected and repeatedly prompted to grade student submissions using the same context that a human teaching assistant (TA) was given.

This paper analyzes 1,172,383 API requests, totaling 33.4 days of active runtime, evaluating model consistency, ability to adhere to specified constraints, and comparison to human-generated grades. The results show various models' inability to consistently grade assignments, albeit with some exceptions. The importance of providing comprehensive context to models was highlighted, as incomplete contexts resulted in worse performance. Other models struggled with longer prompts, delivering less consistent results. Despite disparities between AI-generated and human-assigned grades, the potential for refinement is clear; improved rubrics or selective fine-tuning could enhance model output.

Future work will focus on analyzing models' qualitative justifications for grades, refining rubrics, training on domain-specific datasets, and fine-tuning the highest performing models to potentially improve grading accuracy.

***Index Terms*—Large language model (LLMs), automated assessment tools (AATs), CS1**

I. Introduction

As the demand for computer science education continues, maintaining consistency, accuracy, and speed in assessing student assignments is integral to managing larger courses. Traditional manual grading methods often suffer from subjectivity, inconsistency, and considerable time requirements [14, 26, 30]. In response, Automated Assessment Tools (AATs) have emerged as promising solutions. While originally these systems existed to help instructors ease the task of grading student work [14, 15], educators quickly understood the positive role these systems could have on the student submission process. Researchers began using AATs to aid students in understanding error messages, code decision trees, insecure code, and stylistic errors [8, 38]. Recent systems are capable of hint generation, intelligent tutoring, getting students to start assignments sooner, and personalized feedback [1, 8, 10, 22].

However, AATs come with inherent flaws. From the student's viewpoint, challenges arise from failing test cases due to spelling errors, rigid output matching criteria, inflexible test cases, and demands to adhere to specific formats. Meanwhile, educators often find that these systems only evaluate output, lacking a holistic assessment of students' submissions. Mitigating these issues often demands more effort than directly using human feedback [29, 30]. While large language models (LLMs) are not the sole remedy, they offer an opportunity to integrate elements into existing AAT frameworks, thereby addressing some of these limitations.

LLMs demonstrate proficiency in understanding and generating human-like text, responding to queries, generating code, completing tasks, and engaging in dialogue. Recent research has already highlighted AI's capacity to enhance compiler error messages, conduct static analysis, and provide detail to program error messages [3, 19, 35].

The adoption of LLMs for assessing student submissions is gaining traction, with some educators embracing industry solutions. For example, Gradescope, a for-profit AAT [37], has introduced an AI feature that streamlines the grading process

by grouping similar student responses for efficient collective evaluation. Additionally, initiatives like those at Stanford and Code.org are actively developing AI models to facilitate the grading of Scratch programs [7, 24]. These efforts mark a significant shift towards harnessing AI to enhance the efficiency and effectiveness of educational assessment processes. By employing AI to analyze test case outputs and student code, there is potential to introduce more flexible grading paradigms, complementing the established benefits of traditional methods.

For educators seeking to integrate LLMs into their educational systems, a common approach involves incorporating remotely-hosted (RH) solutions like GPT's API. However, this integration can give rise to significant concerns, including those related to privacy, security, and cost. Students and professionals frequently input sensitive data to LLMs without realizing that the models they are interacting with are trained on the prompts they receive, creating important privacy concerns. A code snippet could inadvertently contain sensitive information, such as student names, passwords, or other intellectual property. This oversight can have consequences, as evidenced by prominent instances where proprietary information has been inadvertently revealed as output to users [31].

Price is another limiting factor, with the potential to create gaps between educators who can afford the cutting-edge versions, and those who cannot. AI continues to be a significant financial drain, with companies losing billions to retain users. Pricing of remote solutions may well increase to offset these losses in the near future [12, 33].

Open-source, locally-hosted (OSLH) LLMs present a compelling alternative to RH models. By enabling educators to download and deploy these models on local servers or personal computers, they gain access to a customizable and private environment where requests never leave a private network. This approach not only addresses concerns related to privacy and security but also eliminates the dependency on external providers, mitigating the risk of rising costs associated with remote solutions. Furthermore, by embracing open-source solutions, educators can foster collaboration and innovation within the educational community, ensuring equitable access to advanced AI technologies regardless of financial constraints.

This preliminary study examines how effectively OSLH LLMs can grade student submissions from a midsized CS1 course during the 2023 semester. In this study, we pose three research questions:

RQ1: *Given enough context, are OSLH models able to assess student submissions accurately and consistently?*

RQ2: *How does the complexity and length of a programming assignment affect OSLH LLM's grading performance?*

RQ3: *Which of the OSLH models perform well when tested with real world grading data?*

II. Related Work

Recent research has explored various facets of using LLMs for grading in educational settings, particularly in computer science courses. These studies highlight both the potential and limitations of AI-assisted grading systems.

Enhancement of error message interpretation and automated fixes in programming assignments are a significant area of inquiry. Studies show that providing context to LLMs improves their ability to produce correct explanations and specific fixes for programming errors, thereby enhancing the utility of AI in educational settings for introductory courses [35]. AI has also been shown to be able to solve the complex problems of advanced programming courses. OpenAI's Codex has been tested in Computer Science 2 (CS2) exams and demonstrated a high capacity for solving complex programming tasks, positioning it favorably among the top quartile of student performers [9]. By providing the appropriate context to the models, we aim to significantly improve the likelihood of consistent and accurate model outputs.

The impact of AI on grading practices has been a significant area of focus. LLMs like GPT-4 generally perform well in grading tasks, achieving an overall accuracy of 75%, although they struggle to accurately identify failing submissions [25]. The application of Chain-of-Thought (CoT) prompts particularly with contextual instructions and scoring rubrics improve the scoring performance of AI grading, further highlighting the importance of providing context [18].

The integration of AI in educational settings presents several challenges. AATs have shown effectiveness in verifying functional correctness during lab sessions, yet they fall short in assessing comprehensive criteria during tests and exams [20]. Such tools also tend to be more effective for high-achieving students and less dependable for those with lower academic performance. Furthermore, automatic assessment is known to struggle to recognize contradictions and is less effective for students with poor language skills [36]. Additionally, the quality of outputs from large language models (LLMs) is sometimes inconsistent. For example, in a high school algebra class, 30% of ChatGPT-generated hints were rejected due to incorrect answers or flawed solution steps [28].

The perception of fairness and accuracy in AI grading has also been explored. Students are skeptical of this technology, believing that AI-graded questions are statistically significantly less accurate than other grading methods [2]. These perceptions highlight the need for transparency and educational initiatives to build trust and understanding of AI's capabilities and limitations in academic settings.

Theoretical research has explored this topic as well. One scenario discussed is an overburdened professor considering an AI grading service to help manage their heavy workload [16]. The benefits claimed are time savings and consistent feedback, while the pitfalls include privacy, legal, ethical, and quality concerns. A core motivation for our research is to identify the most effective strategies for integrating a successful model into an existing AAT framework, with the goal of utilizing AI to further alleviate instructor workloads.

III. Method

A. Environment and Model Selection

Our testing infrastructure was comprised of an NVIDIA RTX 4090 GPU with 24 Gigabytes of GDDR6X VRAM, a i9-

9900K CPU, and 32 Gigabytes of RAM, with a total machine cost of $2600 USD based on market prices in early 2024. This configuration was chosen to mirror the hardware accessible to typical small to midsize university departments.

The software tool Ollama [27] was used to simplify the local management of LLMs, providing a vast selection of pre-configured models optimized for various AI tasks. Even with this tool, identifying ideal hardware limits for models is not straightforward. The Ollama documentation provides some insight, noting: "You should have at least 8 GB of RAM available to run the 7B [billion parameters] models, 16 GB to run the 13B models, and 32 GB to run the 33B models." [27] However, this does not take into consideration GPU constraints, stability concerns, or increased response times, which could occur if not paired with an environment equipped with sufficient resources. GPU's have a finite memory bandwidth, often a bottleneck when running or training larger models [23]. This block results in increased idle times, throttling, degradation of output, and increased crash rates [23]. Given the constraints of a consumer-grade 4090, we restricted the list of OSLHs to those with parameters $\leq$ 13B.

Although the NVIDIA RTX 4090 leverages the latest tensor cores, offering substantial memory bandwidth and computational power, it is still inadequate for managing models above 13B. The hardware demands to run such larger models are often enormous and far exceed a typical department's instructional budget. Furthermore, by selecting models within this parameter range, we can minimize instability, crashes, and increase consistency. By focusing on models within the 13B threshold, we aim to balance the trade-offs between performance and resource availability.

Further expanding on Ollama, this lightweight and extensible framework offers an API for creating, running, and managing models locally. Its pre-built library facilitates easy integration into various applications, such as API requests via `curl`, or Python libraries such as Langchain [17].

With the proliferation of LLM's, there has been a significant increase in the number of OSLH models available. To evaluate the best candidate models for this experiment, we relied on two main indicators. The first indicator used was downloads from the Ollama website, as well as community feedback on model stability. Numerous OSLH models make lofty claims of performance, yet often proved to be incorrectly ported to the Ollama platform, preventing API requests from being processed correctly.

The second indicator that was used was the Open LLM Leaderboard on Hugging Face [4]. The Open LLM Leaderboard was designed to navigate the rapidly evolving landscape of LLMs and chatbots, providing clarity on the genuine progress within the open-source community and identifying the current state-of-the-art models. This platform evaluates models based on seven key benchmarks within the EleutherAI Language Model Evaluation Harness [11], aiming to test a variety of reasoning and general knowledge across different fields. The benchmarks include the AI2 Reasoning Challenge [5], HellaSwag [39], MMLU [13], TruthfulQA [21], Winogrande [34], and GSM8k [6], with evaluations in both 0-shot and few-shot settings to assess models' reasoning, inference, and knowledge capabilities.

Detailed results and logs of these evaluations are publicly accessible, with numerical outcomes available on the Hugging Face dataset page, and model details and community queries updated regularly. To ensure replicability, the Open LLM Leaderboard specifies the use of the EleutherAI Harness with detailed instructions for reproducing results. Models are categorized by their training approach, including pre-trained, fine-tuned, chat models, and base merges, with icons indicating their classification. For our study, we chose exclusively pre-trained models easily available through Ollama.

Model Name	**Size (B)**	**Description**
Codellama	7	Meta model specialized for code generation and understanding.
Llama2	7, 13	Meta general-purpose language model designed for efficient inference, optimized for chat and dialogue applications.
Mistral	7	Mistal AI general-purpose model designed for high performance with an efficient interface.
Orca 2	7	Microsoft model fine-tuned on Llama 2 for enhanced reasoning abilities using synthetic data.
Phi2	2.7	Microsoft model adept at common-sense reasoning and language understanding, performs well in QA, chat, and code.
Qwen	14	Alibaba Cloud model pre-trained on web texts, books, and code, excels in common-sense, reasoning, code, and mathematics.
Solar	10.7	Upstage AI model built on depth up-Scaling technique for single-turn conversation.
Tinyllama	1.1	StatNLP Research Group model optimized for low computation and memory footprint apps.
Yarn-Llama2	7	Nous Research extension of Llama 2 specialized in processing extended context sizes.

TABLE I: Summary of Selected Language Models

B. Experiment design

To answer our research questions, we designed a study in three phases:

1) *A stability assessment phase for OSLH LLMs using deliberately incorrect code solutions,*
2) *Using promising OSLH LLMs to grade student submissions from a CS1 course,* and
3) *Comparing the grading performance of OSLH LLMs against actual grades assigned by human teaching assistants (TAs) or instructors.*

C. Stability Assessment (Phase 1)

The first phase of this experiment was to assess if models were capable of consistently producing usable output, and stable enough to maintain thousands of sequential API requests with varying token lengths.

To manage this, we tasked a model with generating 300 diverse incorrect solutions for the first CS1 assignment, encompassing errors in both semantics and logic. Each submission underwent testing against the assignments test case suite producing results accordingly.

The prompt to the models was structured as follows:

- *Code: The generated incorrect code solution.*
- *Assignment Description: The instructions provided to the students and the expected output required for the project.*
- *Test Case Results: The students' submissions are formatted as TAP (Test Anything Protocol) files. These files provide detailed test results by running the submissions against the test cases of a known working assignment. This process highlights any discrepancies between the expected and actual results. (see Figure 1)*
- *Grading Rubric: The guidelines for assigning points. The same rubric was given for every assignment, and it is the same one that TAs used when grading.*
- *Expected output: We enforced JSON as the response format in the LangChain model caller. This format, which includes "Grade": (X/100), "Reasons": [...].*

```
not ok 1 - Example test name
  ---
  description: 'A simple example'
  got: 'actual output'
  expected: 'expected output'
  diff:
    - '< expected output'
    - '> actual output'
  ...
```

Fig. 1: Example of a Test Anything Protocol (TAP) Formatted Output, Demonstrating a Comparison of Linux `Diff` Outputs.

To ensure ample data for analysis, each LLM received 50 identical prompts per submission, with responses promptly checked for validity. At this stage failure was assessed based on three criteria: a null check, adherence to requested JSON format, and a naive check for indications of a grade using text processing algorithms. Any poorly formatted outputs triggered a logging of failure and repetition of the process. These types of failures are defined as a pre-processing error, ϕ_{pre}.

This protocol aimed to mitigate potential output irregularities due to processing issues. If a model could not provide an appropriate response after ten attempts, it was marked as failure, and evaluation proceeded to the next iteration. This approach sought to balance efficiency with accuracy and reliability. Under ideal circumstances, with no model skips, this would result in 15,000 LLM output runs.

These failure checks, while able to eliminate basic mistakes, were not able to catch all of the varied output possibilities. Thus, another set of assessments was developed to process API responses that passed the initial failure checks but still contained unacceptable data. We extracted grades from the model's output accommodating variations in formatting such as different prefixes, optional quotes, and a variety of separators. This algorithm handles numeric grades, including decimals and fractions, and discards non-digit characters. It also ensures adherence to common punctuation conventions. Despite the careful design of the extraction algorithm, there remained instances where grades still could not be extracted from the LLM output. These types of failures are defined as a post-processing error, ϕ_{post}:

- ϕ_{range} *- Extracted grades do not fall within the acceptable range of 5 to 100. Occasionally, outputs may mistakenly identify a student's ID number as a grade or display invalid grades such as 0 or 1.*
- ϕ_{diff} *- When multiple grades appear in a single response, the algorithm assesses whether the difference between the highest and lowest grade exceeds one.*
- ϕ_{format} *- The required format for entering grades is 'X/100'. Often, the output incorrectly presents only the placeholder text 'X/100' without an actual grade, rendering it invalid.*
- ϕ_{empty} *- This error message appears when the system is unable to identify any valid grades in the input provided.*

D. Assessing Student Submissions (Phase 2)

For phase 2 of this experiment, the top six performing models from phase 1 were given the task of grading student submissions from a CS1 course. The CS1 course, with 124 students and nine weekly programming assignments, used an established AAT [10], which stored student data for this project. To enhance the reliability of findings from this phase, each model repeated 100 iterations per submission. Under ideal circumstances, where models did not skip any iterations, this would yield a total of 111,600 LLM output runs.

The prompt for Phase 2 remained consistent with Phase 1, consisting of the student code, assignment description, and test case results tailored to each specific submission and assignment. As in phase 1, each iteration was required to adhere to the output format: "Grade": (X/100), "Reasons": [...].

In addition, Phase 2 maintained the robust ϕ_{pre} and ϕ_{post} detection mechanisms, allowing for up to ten repeat requests and comprehensive failure logging, as outlined in the preceding section. These measures ensured thorough scrutiny of model responses and enabled the identification of errors encountered during the grading process.

IV. Results

A. Phase 1: Trial Phase

In this experiment, the aim was to compare the performance of various models through their ability to predict grades accurately. In phase 1 of the experiment, two key metrics were used: success rate and the pooled standard deviation. *Success rate* (π) measures how often a grade can be extracted from a given model's 15K runs (300 deliberately wrong code submissions, 50 iterations each).

Meanwhile, the *pooled standard deviation* (σ_p) measures the consistency of model outputs across all submissions and iterations. This was done by calculating the σ for each model's 50 iterations on a particular submission, recording the number of successful extractions for each iteration, and accounting for the total number of σ calculated. σ_p ensures more weight is given to the iterations with larger extraction rate, and thus higher sample size. This calculation reflects the variability of grades across all submissions for a specific model.

A composite score for each model was then calculated by weighting these metrics, $C = 0.7 \times \pi + 0.3 \times 1/\tilde{\sigma}_p$. The π carries more weight because it is essential for a model to reliably output a grade before evaluating the grade's consistency. Without a high π, a meaningful analysis of grade variability cannot occur, thus making this metric crucial. Only 30% of inverted, normalized, pooled standard deviation ($1/\tilde{\sigma}_p$) is contributed to the score because it relies on having sufficient grade data in the first place. This balanced approach helps identify the models that perform best in both accuracy and consistency.

TABLE II: Phase 1 Model Statistics and Composite Scores

Model	Success (%) (π)	Pooled Std. Dev. (σ_p)	Composite Score
Solar	100.0	7.7	0.9875
Llama2-7b	90.0	8.8	0.8628
Llama2-13b	90.9	11.4	0.8268
Mistral	89.4	12.2	0.7973
Codellama-7b	91.5	15.3	0.7653
Phi	66.7	10.3	0.5896
Orca2-7b	67.2	14.0	0.5301
Yarn-llama2	66.4	23.4	0.3581
Qwen-14b	33.8	6.9	0.3000
Codellama-13b	54.6	21.9	0.2589
Tinyllama	50.3	24.1	0.1753

The purpose of generating this composite score was to select the top six models for the phase 2 of the study. These models needed a simple metric to evaluate how well each model would do in the next phase, where they would do this same task on actual code submissions from a CS1 class.

B. Phase 2: Testing Student Written Code

After choosing the models to test in phase 2, analysis was started on CS1 student work that would be fed into each model. The prompts and subsequent data were divided into two categories: perfect grades (assignments that received a 100) and non-perfect grades (assignments scored below 100). This separation serves two purposes. Firstly, non-perfect grades provide more information to the model because the input prompts included the TAP difference file, whereas if there are no differences, the input simply states, "Student's code passed all tests," resulting in shorter input statements. Secondly, as mentioned previously, in a real-world application, there would be minimal need to grade perfect submissions since passing all test cases is a clear indicator of success.

Examining the average token count per assignment, divided into perfect and non-perfect categories, indicates which models perform better with longer prompt token counts. Figure 2 illustrates this trend, revealing that input statements for non-perfect grades tend to be lengthier compared to those for perfect grades, due to the test result having more tokens. As assignments progress, both the complexity of the homework prompt and the resultant student code increases. The gap between perfect and non-perfect submissions also widens, with increasingly complex testing results.

Looking at the total amount of ϕ_{pre}, figure 3 reveals that Assignment 9 ("A9") had the highest frequency of ϕ_{pre},

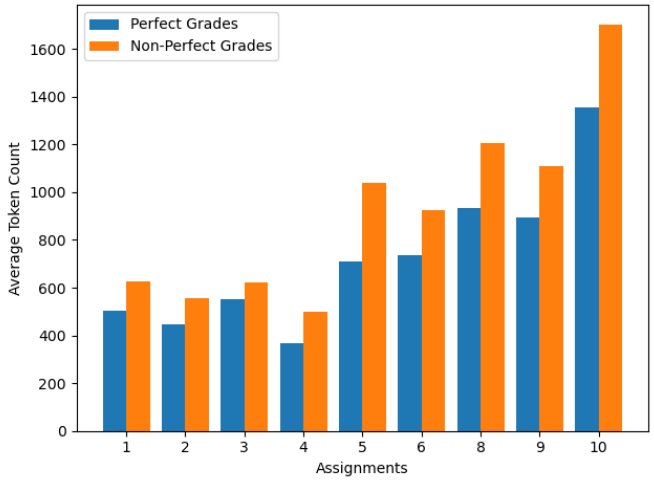

Fig. 2: Average Input Token Count for Each Assignment.

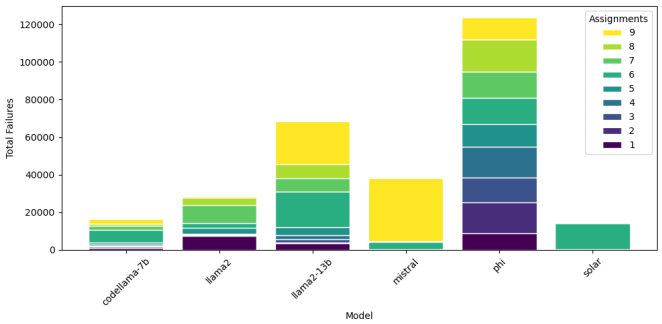

Fig. 3: ϕ_{pre} per Model, per Assignment.

which as previously stated include checks for empty responses, invalid output structure via strict JSON format, and a quick check for an indication of a grade. A9 accounted for 24.9% of all ϕ_{pre}. Close behind, A6 accounted for 20.7%. The models that experienced high rates of ϕ_{pre} include Phi (accounting for 42.7% of all total ϕ_{pre}), Llama2-13b (23.6%), and Mistral (13.2%) while Solar (5.0%), Codellama (5.7%), and Llama2 (9.8%) had low rates of these errors.

Showing the total amounts of ϕ_{post}, the data from figure 4 indicates that Llama2, Solar, and Mistral having experienced the lowest levels of ϕ_{post} among the models, accounting for 3.8%, 7.0%, and 12.3% of the total ϕ_{post} among all models. Phi, Llama3-13b, and Codellama on the other hand recorded high amounts of these failures, accounting for 41.5%, 20.2%, and 15.4%. Notably, A9 (accounting for 15.4% of errors among all assignments) had the second most ϕ_{post}, while A6 (32.2%) had by far the highest overall ϕ_{post}.

The predominant error type that emerged in testing was *no grade found*, (ϕ_{empty} = 50.2%). As shown in Figure 5, the distribution of error types varied across different models. For example, Llama2-13b frequently generated template code errors ($\phi_{format} = 56.4\%$), while Mistral often produced grades outside the range 5-100 ($\phi_{range} = 48\%$).

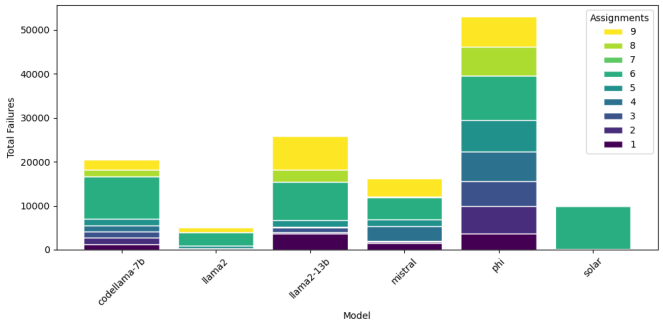

Fig. 4: Total ϕ_{post} per Model, per Assignment.

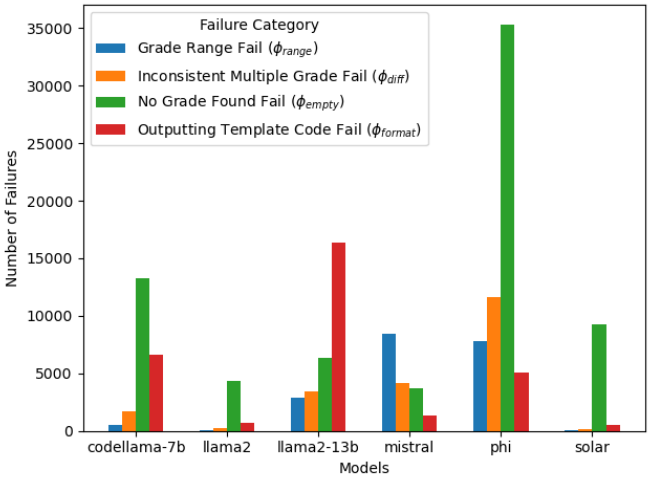

Fig. 5: Post-Processing Errors (ϕ_{post}) by Failure Category.

An analysis of the frequency with which grades were extracted for each assignment (excluding ϕ_{pre}) reveals some interesting results. As depicted in Figure 6, Solar achieved the highest π across all assignments, with the lowest π being 98.7%, except for A6. In this case, Solar only managed to extract the grade 14.5% of the time. Meanwhile, Llama2 showcased its adaptability by achieving the highest model π of 72.9% on A6, as well as having a high π on the other assignments. On the other hand, Phi struggled to extract grades consistently, with its best performance being a 71.6% π on A1 and goes down as low as 10.3%. Mistral and Codellama did fairly well, but A6 and A9 hurt their performance.

Given ϕ_{pre}, ϕ_{post} and successful extractions (as shown in Figure 7), Solar had the best π with 79.8%, followed close by Llama2 with 75.1%, and then by Codellama and Mistral with 68.9% and 61.3% respectively. Llama2-13b had only 44.2%, while Phi had a mere 20.5%.

In total, there were 497,488 successful extractions, 289,386 ϕ_{pre}, and 144,089 ϕ_{post}, bringing the total number of queries processed by these LLMs in Phase 2 to 930,963.

To assess the success of this phase, we measure how consistently the model assigned a particular grade, partnered with accuracy in reflecting the actual grade. To measure

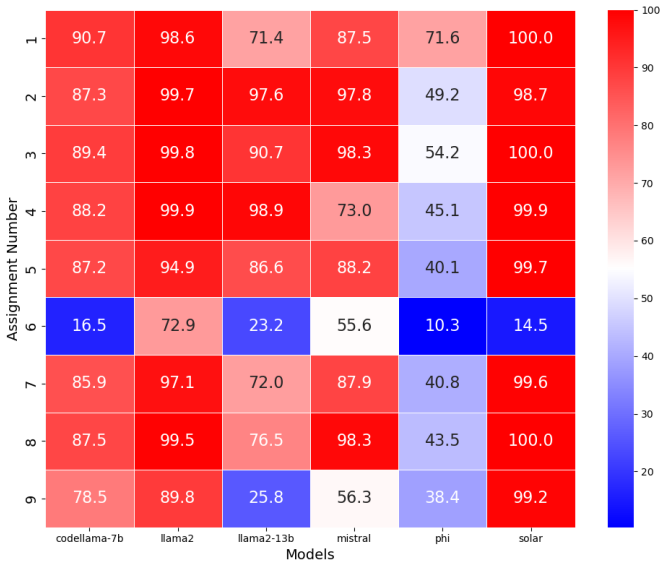

Fig. 6: Grade Extraction Heatmap (Excluding ϕ_{pre}): π per Model, per Assignment (Red = High Success).

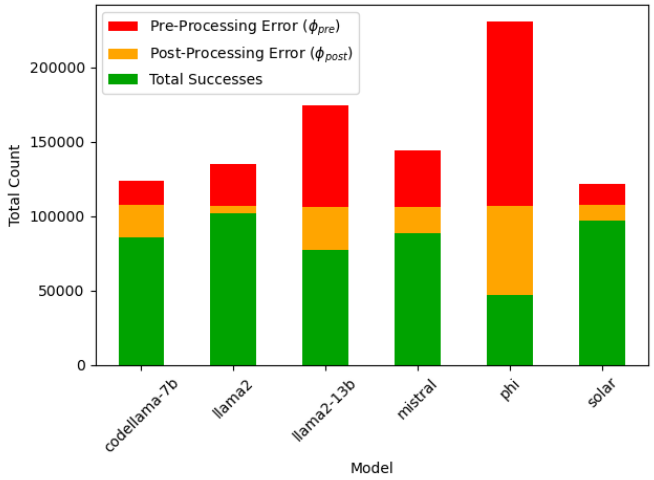

Fig. 7: ϕ_{pre}, ϕ_{post}, and Successful Extractions.

consistency, the σ_p was calculated for each assignment, for each model, and for each assignment-model combination. The assignment-model combination statistic was then visualized in a heatmap (Figure 8). The scale of 0-10 was chosen to represent the range of σ_p, indicating a practical upper limit for variability in grade predictions, where a higher value reflects greater inconsistency among the model's predictions.

The heatmaps indicate that the models were unable to assign a consistent grade. Solar had the lowest σ_p for non-perfect scores at 6.6 overall, while Llama2 had 6.9. Every other model had a σ_p above 10. The maximum σ_p was A6 at 14.7.

The grades predicted by the model were compared against the actual human-generated grades awarded in the course. The average weighted predicted grade was calculated for each student for each assignment. These predicted grades were

Fig. 8: Non-perfect grade σ_p per Model, per Assignment (Red = Success, 0 Inconsistency).

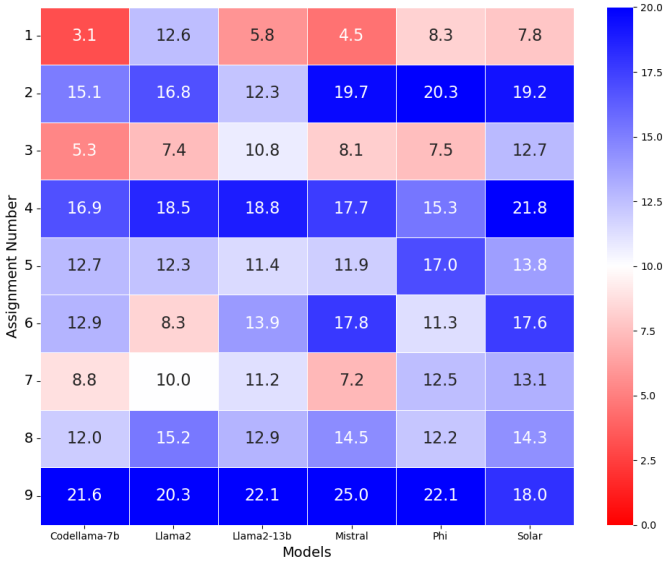

Fig. 9: Non-Perfect Grade Average of Absolute Value Difference per Model, per Assignment (Red = Success, 0 Difference)

then directly compared with the actual grades received. The absolute value was taken on the difference to not reward models for having a scattershot approach (having numbers both above and below the actual grade). These differences are visually represented in a matrix format, correlating assignment numbers with model predictions, as shown in Figure 9.

Figure 9 demonstrates how every model is not consistently aligned with the grades given by TAs during the semester. Codellama marginally did the best at an average weighted absolute value difference of 12.0. Llama2 and Llama2-13b had similar numbers of 13.3 and 13.5. Phi and Mistral were at 14.0 and Solar was at 15.3. A9 experienced the highest average absolute difference at 21.5.

To further this point, two more metrics were calculated. For each grade produced for a particular assignment, the output was compared with the actual grade. It was subsequently assessed whether it fell within a margin of 5 or 10 points of the grade. These results show that Mistral was the best performing model at this metric with 40.5% of all grades within 5 points, and 55.8% of all grades within 10 points.

V. Discussion and Conclusion

The findings indicate that of the tested OSLHs with parameters $\leq$ 13B, most models are inconsistent in assigning numeric grades. Solar, Llama2, and Mistral exhibited the most consistency of the tested models, but were still not very consistent. Although A6 is an outlier for all models, it was the only assignment including a reference to a diagram that was integral to comprehension.

This reinforces previous research indicating that providing more context to models results in more accurate outputs [35]. What is equally interesting is it provides insights into what information models are using to render decisions. Unlike traditional input/output assessment, models seem to be taking a holistic approach, as A6 had the same type of test case results.

Each model had different strengths and weaknesses. For instance, some models had more difficulty when not given the full context. In Solar's case, most failures occurred due to the issues identified with A6. Similarly, prompt length affected some models more than others. For example, Mistral was particularly sensitive to longer prompts, especially in ϕ_{pre}. In contrast, Llama2 handled these extreme edge cases more effectively, achieving the highest π in A6 and the second-highest in A9 during Phase 2.

Despite the disparities between the actual grades assigned and those assigned by TAs for a given assignment, this research underscores the potential of certain models for refinement and further investigation.

A. Threats to Validity

The benchmarks used to select the models had limitations. The Open LLM Leaderboard benchmarks, not tailored for grading CS1 assignments, may not fully reflect real-world performance. In addition, the downloads indicator could have been influenced by marketing and user familiarity. However, using both indicators helped to mitigate these limitations.

The findings presented are influenced by the fundamental reality that the performance of models is inherently limited by the hardware they operate on [40]. While there is a possibility that specific models, notably Llama2-13B, could achieve better performance with more advanced hardware, all tested models operated within the recommended hardware specifications outlined by Ollama. Moreover, while investing in a significantly more expensive testing platform is conceivable, such a decision would directly contradict one of the key motivations behind the importance of evaluating self-hosted models: cost-effectiveness. This consideration is especially

pertinent in the context of universities, particularly amidst periods of downsizing and budgetary constraints [32]. While leveraging RH platforms may appear cost-effective in the short term for small prompt sizes, it raises significant security and ethical concerns, rendering it an impractical option for many universities. In phase one, when utilizing incorrect solutions based on A1, we initially explored generating responses using GPT-4 as another point of comparison. However, after 2,104 API calls, our API bill cost $123.73, averaging around $0.064 per API request for one of our shortest prompt length submissions. Even under ideal circumstances, assuming no rejected results in phase one (which was not the case in practice), this would lead to an expected average bill of $952.5 for 15,000 API requests. Extrapolating this to phase two with 111,600 API calls, where prompt lengths were often double in size, yields a conservative lower bound estimate of $7,087.

Our findings also extend to an incomplete dataset, where the N values associated with iterations lacked uniformity. While one can continuously prompt models until feedback is consistently formatted, there is no guarantee of termination. Opting instead to assign a failure rate to models after 10 repeated failed outputs helps underscore inconsistencies in the models' capacity to adhere to specific output formats, a critical aspect to consider in workflows necessitating swift parsing of model outputs. This is a necessary reality if you intend to integrate this into any existing AAT workflow.

The study, conducted in a CS1 course at our university using our custom AAT, may yield different results at other institutions due to varying factors such as hardware, software, institutional priorities, or the teaching of different CS courses.

B. Conclusion

To address the research questions outlined in our introduction, we conducted an extensive analysis on 1,172,383 API requests, totaling 33.4 days of active runtime. Our findings indicate that there is inconsistency in providing grades and difficulty in aligning with human-assigned grades. However, these issues improve when more context is provided [RQ1]. As assignment complexity increases (resulting in longer prompts), some models maintain π, while others decline. Solar marginally improved as prompt complexity increased [RQ2].

Two models—Solar, and Llama2—consistently delivered output with relatively low failure rates, each excelling in specific metrics. Llama2 performed well with low context input statements and matched the TA-generated grade more than other models. In contrast, Solar achieved a higher π, showed the lowest σ_p, and was more effective with longer inputs. Definitively identifying the best model depends on what one weighs as the more important criteria [RQ3].

VI. Future Work

Our experimental prompt requests both a grade and a justification for the assigned grade. The work presented here focuses on evaluating the models' ability to provide consistent and accurate grades. If a model cannot accurately or consistently assign a grade to a submission, it is difficult to have confidence in its reasoning for point deductions. However, with data showing consistent model output, even if the result differs from the human-generated grade, it is worth examining why models claim to have deducted points. The analysis of these qualitative results is already in progress.

Further exploration could focus on refining rubrics, as a more detailed and specific grading rubric could mitigate subjectivity and enhance consistency, not only among LLMs but also among human graders. Moreover, employing custom models, ensemble models, or training on domain-specific datasets for CS1 may enhance their grading capabilities. For top-performing models, fine-tuning them to better comprehend nuances and to weigh particular aspects of a student's submission could lead to even more promising results.

Another crucial consideration is refining the presented evaluation frameworks to assess grading quality more accurately. Currently, the grading was based solely on the score provided by the TA. This process could be enhanced by incorporating inter-rater reliability and conducting error analysis to identify common mistakes. With better performance metrics, the grading process can be rigorously evaluated and improved.

These models as currently constituted should not determine final student grades. Instead, these models should be used to assist TAs by providing additional feedback, supporting consistent and efficient grading while ensuring that human evaluators make the final decision.

References

[1] P. Antonucci, C. Estler, D. Nikolić, et al. An incremental hint system for automated programming assignments. In *ITiCSE '15*, pp.320–325, 2015. ACM. DOI 10.1145/2729094.2742607

[2] S. Azad, B. Chen, M. Fowler, et al. Strategies for deploying unreliable AI graders in high-transparency high-stakes exams. In *AI in Education*, pp.16–28, 2020. Springer. DOI 10.1007/978-3-030-52237-7_2

[3] R. Balse, V. Kumar, P. Prasad, et al. Evaluating the quality of LLM-generated explanations for logical errors in CS1 student programs. In *COMPUTE '23,* pp.49—54, 2023. ACM. DOI 10.1145/3627217.3627233

[4] E. Beeching, C. Fourrier, N. Habib, et al. Open LLM leaderboard, 2023. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

[5] P. Clark, I. Cowhey, O. Etzioni, et al. Think you have solved question answering? Try ARC, the AI2 reasoning challenge, 2018. DOI arXiv.1803.05457

[6] K. Cobbe, V. Kosaraju, M. Bavarian, et al. Training verifiers to solve math word problems, 2021. DOI 10.48550/arXiv.2110.14168

[7] Code.org. AI teaching assistant, 2024. Accessed on 2024-05-12, https://code.org/ai/teaching-assistant.

[8] T. Crow, A. Luxton-Reilly, and B. Wuensche. Intelligent tutoring systems for programming education: a systematic review. In *ACE '18*, pp.53—62, 2018. ACM. DOI 10.1145/3160489.3160492

[9] J. Finnie-Ansley, P. Denny, A. Luxton-Reilly, E. Santos, et al. My AI wants to know if this will be on the exam: Testing OpenAI's codex on CS2 programming exercises. In *ACE '23*, pp.97–104 2023. ACM. DOI 10.1145/3576123.3576134

[10] J. Forden, A. Gebhard, and D. Brylow. Experiences with TA-Bot in CS1. In *CompEd 2023*, pp.57-–63, 2023. ACM. DOI 10.1145/3576882.3617930

[11] L. Gao, J. Tow, S. Biderman, et al. A framework for few-shot language model evaluation, Sep 2021. DOI 10.5281/zenodo.5371629

[12] Gizmodo. Microsoft's AI copilot is basically OpenAI's GPT-3.5, 2022. Accessed on 2024-05-12, https://gizmodo.com/github-copilot-ai-microsoft-openai-chatgpt-1850915549

[13] D. Hendrycks, C. Burns, S. Basart, et al. Measuring massive multitask language understanding, 2020. DOI 10.48550/arXiv.2009.03300

[14] J. Hollingsworth. Automatic graders for programming classes. *Commun. ACM*, 3(10):528–529, Oct 1960. DOI 10.1145/367415.367422

[15] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Koli Calling '10*, pp.86–93, 2010. ACM. DOI 10.1145/1930464.1930480

[16] R. Kumar. Faculty members' use of artificial intelligence to grade student papers: a case of implications. *International Journal for Educational Integrity*, 19(1):9, 2023. DOI 10.1007/s40979-023-00130-7

[17] LangChain. Ollama Documentation. 2024. https://python.langchain.com/docs/integrations/llms/ollama

[18] G. Lee, E. Latif, X. Wu, et al. Applying large language models and chain-of-thought for automatic scoring. *Computers and Education: Artificial Intelligence*, 6:100213, 2024. DOI 10.1016/j.caeai.2024.100213

[19] H. Li, Y. Hao, Y. Zhai, and Z. Qian. Enhancing static analysis for practical bug detection: An LLM-integrated approach. *Proc. ACM Program. Lang.*, 8(OOPSLA1), Apr 2024. DOI 10.1145/3649828

[20] J. Liebenberg and V. Pieterse. Investigating the feasibility of automatic assessment of programming tasks. *J. Information Technology Education: Innovations in Practice* 17:201–223, 2018. DOI 10.28945/4150

[21] S. Lin, J. Hilton, and O. Evans. TruthfulQA: Measuring how models mimic human falsehoods. In *ACL '22*, pp.3214–3252, 2022. DOI 10.18653/v1/2022.acl-long.229

[22] M. Messer, N. Brown, M. Kölling, and M. Shi. Automated grading and feedback tools for programming education: A systematic review. *ACM Trans. Comput. Educ.*, 24(1), Feb 2024. DOI 10.1145/3636515

[23] D. Narayanan, M. Shoeybi, J. Casper, et al. Efficient large-scale language model training on GPU clusters using megatron-LM. In *SC '21*, 2021. ACM. DOI 10.1145/3458817.3476209

[24] A. Nie, E. Brunskill, and C. Piech. Grading complex interactive coding programs with reinforcement learning, 2022. https://ai.stanford.edu/blog/play-to-grade/

[25] F. Nilsson and J. Tuvstedt. GPT-4 as an automatic grader: The accuracy of grades set by GPT-4 on introductory programming assignments. DiVA, id: diva2:1779778

[26] S. Nutbrown and C. Higgins. Measuring the impact of high quality instant feedback on learning. *Practitioner Research In Higher Education*, 10(1):130–139, 2016. http://files.eric.ed.gov/fulltext/EJ1129863.pdf

[27] Ollama. Ollama GitHub Repository. https://github.com/ollama/ollama, 2024.

[28] Z. Pardos and S. Bhandari. Learning gain differences between ChatGPT and human tutor generated algebra hints. DOI arXiv.2302.06871

[29] V. Pieterse. Automated assessment of programming assignments. In *CSERC '13*, pp.45–56, Heerlen, NLD, 2013. DOI 10.5555/2541917.2541921

[30] V. Pieterse and C. Stallmann. Managing a large tertiary computer science class. In *CSERC '14*, pp.79–90, 2014. ACM. DOI 10.1145/2691352.2691359

[31] L. Maddison. Samsung workers made a major error by using ChatGPT. https://www.techradar.com/news/samsung-workers-leaked-company-secrets-by-using-chatgpt, 2023.

[32] Hanover Research. 2024 trends in higher education. https://www.hanoverresearch.com/reports-and-briefs/2024-trends-in-higher-education/

[33] Reuters. AI companies lose $190 billion in market cap after alphabet, microsoft report, Jan 31 2024. Accessed on 2024-05-12, https://www.reuters.com/technology/ai-companies-lose-190-billion-market-cap-after-alphabet-microsoft-report-2024-01-31/

[34] K. Sakaguchi, R. Bras, C. Bhagavatula, and Y. Choi. WinoGrande: An adversarial winograd schema challenge at scale. DOI arXiv.1907.10641

[35] E. Santos, P. Prasad, and B. Becker. Always provide context: The effects of code context on programming error message enhancement. In *CompEd 2023*, pp.147–153, 2023. ACM. DOI 10.1145/3576882.3617909

[36] J. Schneider, B. Schenk, C. Niklaus, et al. Towards LLM-based autograding for short textual answers. DOI arXiv.2309.11508

[37] A. Singh, S. Karayev, K. Gutowski, et al. Gradescope: A fast, flexible, and fair system for scalable assessment of handwritten work. In *L@S '17*, pp.81–88, 2017. ACM. DOI 10.1145/3051457.3051466

[38] D. Souza, K. Felizardo, and E. Barbosa. A systematic literature review of assessment tools for programming assignments. In *CSEET '16*, pp.147–156, 2016. DOI 10.1109/CSEET.2016.48

[39] R. Zellers, A. Holtzman, Y. Bisk, et al. HellaSwag: Can a machine really finish your sentence? In *ACL '19*, pp.4791–4800, 2019. DOI 10.18653/v1/P19-1472

[40] H. Zhang, A. Ning, R. Prabhakar, et al. A hardware evaluation framework for large language model inference. DOI arXiv.2312.03134