Toward Pedagogically Effective AI for Introductory Computer Science

Michael Wu
Prabal Dutta, Ed.
Emma Pierson, Ed.

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2025-119
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-119.html

May 16, 2025

Toward Pedagogically Effective AI for Introductory Computer Science

by Michael Wu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Prabal K Dutta

Professor Prabal Dutta
Research Advisor

May 16, 2025

(Date)

* * * * * * *

EP

Professor Emma Pierson
Second Reader

May 16, 2025

(Date)

Toward Pedagogically Effective AI for Introductory Computer Science

by

Michael Wu

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Prabal Dutta, Research Advisor
Professor Emma Pierson, Second Reader

Spring 2025

Toward Pedagogically Effective AI for Introductory Computer Science

Abstract

Toward Pedagogically Effective AI for Introductory Computer Science

by

Michael Wu

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Prabal Dutta, Research Advisor

Professor Emma Pierson, Second Reader

The rise of large language models (LLMs) like ChatGPT has begun to revolutionize the educational landscape. Specifically in computer science, this has led to the emergence of LLM-based tutors intended to guide students through the learning process, much like how a human instructor would. However, these systems still face major challenges: hallucination and inaccuracy can mislead students, and over-helping can stifle independent learning. In this thesis, we investigate different design strategies for building more accurate and pedagogically effective LLM-based tutoring systems. We begin by outlining key design principles aimed at balancing assistance with promoting student learning. We then explore different designs and architectures to improve performance measured against these standards. We ultimately find that the best results come from a dual-agent model combined with structured chunking retrieval-augmented generation (RAG), few-shot prompting, and fine-tuning. We hypothesize that this architecture improves performance by combining the generation agent's reasoning with a verification agent that can catch inaccuracies or pedagogical oversteps before they reach the student. Experimental results demonstrate that these techniques improve accuracy and adherence to pedagogical guidelines, particularly in iterative, multi-turn learning scenarios. However, these gains come at the cost of increased computational overhead, requiring more tokens and two model calls per interaction, which negatively impacts inference time and operational cost. Our findings offer a framework for building tutoring agents that better support student learning and describe the trade-offs inherent in LLM deployment.

To my family

Contents

List of Figures

List of Tables

Acknowledgments

This thesis would not have been possible without the contributions of many people. First and foremost, I would like to express my greatest thanks to Professor Prabal Dutta, who trusted me to take on this research project, and to Professor Emma Pierson, who graciously agreed to serve as a Second Reader. I also thank all the amazing researchers at the ACE Lab for introducing me to the field of education research and helping me develop as a fledging researcher. Many thanks go out to JunChang Wang and Lydia Zhang for their ideas and help with this project. Also, my four years at Berkeley were really made by the friends I've met along the way. Thank you for all the memories we've made together, and for the many more we will share in the future. Finally, I thank my family for all their love and support.

Chapter 1

Introduction

With the advent of ChatGPT, many students have started turning to large language models (LLMs) to complete assignments, bypassing the actual learning process [24]. In response, some institutions have restricted or even banned the use of LLMs, citing concerns over declining educational outcomes [1].

However, LLMs also offer powerful opportunities to enhance education. In 1984, educational psychologist Benjamin Bloom published the now seminal paper, The 2 Sigma Problem, which detailed his findings that students who received one-on-one tutoring outperformed their peers learning in traditional classrooms by two standard deviations [5]. Private tutoring, while highly effective, remains out of reach of the vast majority of students. LLMs possess the opportunity to change this. LLMs are adept coders [4, 8, 6], capable of understanding student code [12] and providing feedback [25].

Given the recent rise in interest in computer science, many institutions struggle to meet the growing demand for individualized support [16]– LLMs could help bridge this gap. For students learning through MOOCs, LLMs could provide personalized, instructor-like feedback that is otherwise missing. Indeed, some have chosen to embrace this technology and develop LLM-based tutors to offload instructional workload and deliver fast, individualized help [20, 15, 11]. However, this solution still leaves much to be desired. Many existing applications of LLMs for this purpose still rely on some degree of instructor intervention or limit the nature or scope of questions students can ask.

In this thesis, we implement several versions of AI-powered personal tutors for an introductory computer science course. We document the key design challenges and decisions involved in building effective LLM tutors, focusing on addressing common pitfalls such as over-teaching and inaccuracy. We also briefly apply our design to a university-level accounting course to demonstrate potential for broader applicability. Through this work, we aim to contribute to the growing body of research on best practices for building effective LLM tutors.

Chapter 2

Design Principles

Before creating our tutor bot, we first established a set of principles to guide our design choices.

2.1 Accuracy

First, we want the tutor bot to be as accurate as possible. We define an accurate response as one that does not produce any incorrect or irrelevant information. When an LLM guides students down the wrong path, it can cause significant frustration and lead to hours of unnecessary struggle. Some studies have shown that the reliability of AI feedback on programming assignments can be as low as 50% [7]. Although we expect higher performance given improvements in model quality over the last couple of years, and our focus on more introductory CS topics, the risk remains. Human teachers make mistakes too, but they typically qualify their answers and express uncertainty when appropriate. In contrast, LLMs often state incorrect information with unwarranted confidence, making mistakes harder for students to detect.

2.2 Pedagogical Soundness

Second, we want the tutor bot to be pedagogically sound. This means not directly solving problems for students or violating academic integrity guidelines. Educational research over the years has found that the "struggle" and process of self-discovery are critical for long-term information retention and future problem-solving [23]. If the bot over-explains, it could rob students of that valuable learning opportunity. Ideally, the bot would provide the minimal degree of help necessary for students to make progress. While it is impossible to know that exact amount, a natural heuristic would be to start by giving less information and increasing the depth of assistance with subsequent queries. Furthermore, to help avoid the pitfall of over-explaining, we plan to instruct our bot to avoid direct code generation altogether.

However, even this is not a complete solution. Harvard's CS50 "Duck Debugger"[1]—despite explicit instructions not to perform code generation—was found to generate code solutions in 20% of messages and about 50% of conversations overall [14].

In addition to balancing the depth of queries, a tutor bot must use an appropriate tone, which is critical for learning. Research shows that students learn better in a friendly environment with lots of positive reinforcement [10]. Unfortunately, previous works have found that students often perceive AI feedback as unfriendly and difficult to understand [17]. It will be important to ensure the bot consistently maintains a friendly and clear tone.

2.3 Flexibility

Finally, we want the tutor bot to be flexible enough to handle a wide range of student input. Students should be able to ask conceptual questions, seek help with specific problems, or request clarification on any part of the curriculum. Students have already been able to derive value from various forms of LLM assistance. Tools like SPHERE help instructors rapidly give personalized feedback [21], but cannot provide value in the absence of an instructor. Human-AI approaches like Tutor CoPilot have already been shown to improve student outcomes in K-12 math [22], but again rely on a degree of instructor oversight. Purely student-facing tools like CS 61A bot[2] help students debug homework assignments, but do not support chat-based interaction for non-debugging related questions [20]. Our goal is to create an all-in-one tutoring system that can function independently of an instructor.

2.4 Other Considerations

In addition to the three primary design goals, other factors such as operational cost and response latency can significantly influence the overall effectiveness of the model. Both considerations are important: high costs may restrict access to this type of technology, undermining one of our core motivations, which was to provide personal tutoring to those who might not otherwise afford it. Similarly, excessive latency may frustrate users and deter them from using the tool entirely. However, for the purposes of this study, these factors will come second. In cases where trade-offs arise, we will favor design decisions that enhance accuracy, pedagogical soundness, and flexibility, even at the expense of increased cost or slower response times. We will only make choices to keep costs low and latency down if there is no measurable difference in the primary performance metrics. The ordering of our priorities from most important to least important is as follows: accuracy, pedagogical soundness, flexibility, cost, and latency.

[1] https://cs50.ai/

[2] https://cs61a.org/articles/61a-bot

Chapter 3

Methodology

To maximize the performance of our AI tutor, we considered several design choices, detailed below. First, we needed to choose a class on which to base the bot. We chose UC Berkeley's CS 61A, Berkeley's introductory Computer Science (CS1) course, because it had a large amount of available data, and was familiar. Next, we needed to choose an interface to expose our tool to the user. We chose a chatbot interface because it allows for the most flexible interactions: students can ask conceptual questions or paste code snippets for debugging assistance. Figure 3.1 below shows the user interface.

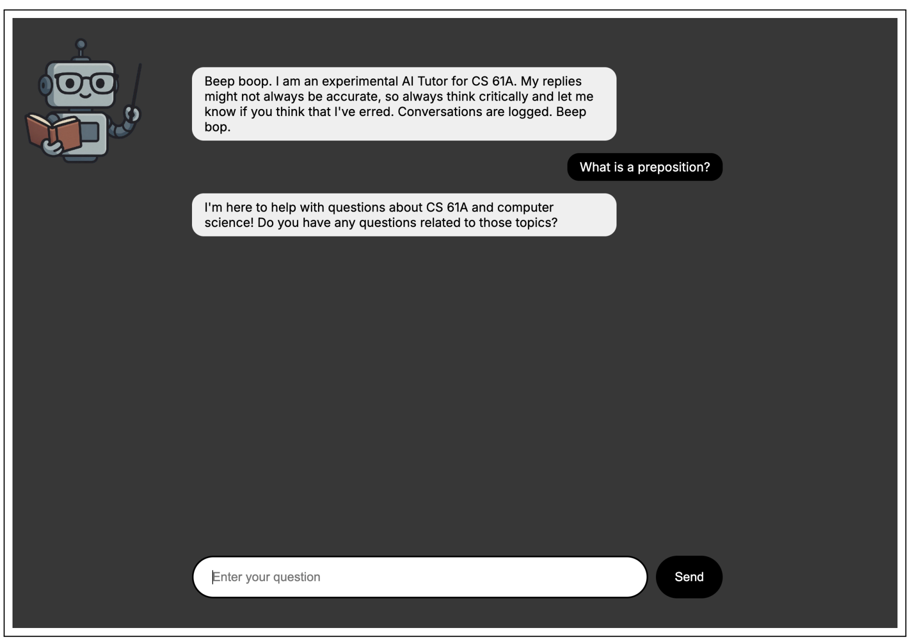


Figure 3.1: UI of the tutoring bot. The student can ask questions or paste code in the text box. Since the bot is instructed to only answer questions about CS 61A or related topics, it refuses the student's question.

Each student query, along with the relevant context retrieved via retrieval-augmented generation, is sent to a GPT-4.1 model. This model is tasked with answering the query to the best of its ability without directly providing solutions. The model's output, along with the original query, is then passed to a second, smaller model (GPT-4o Mini) responsible for verifying the response's correctness and adherence to pedagogical principles. Only the output of the second model is returned to the student. The full system architecture is shown in Figure 3.2 below.

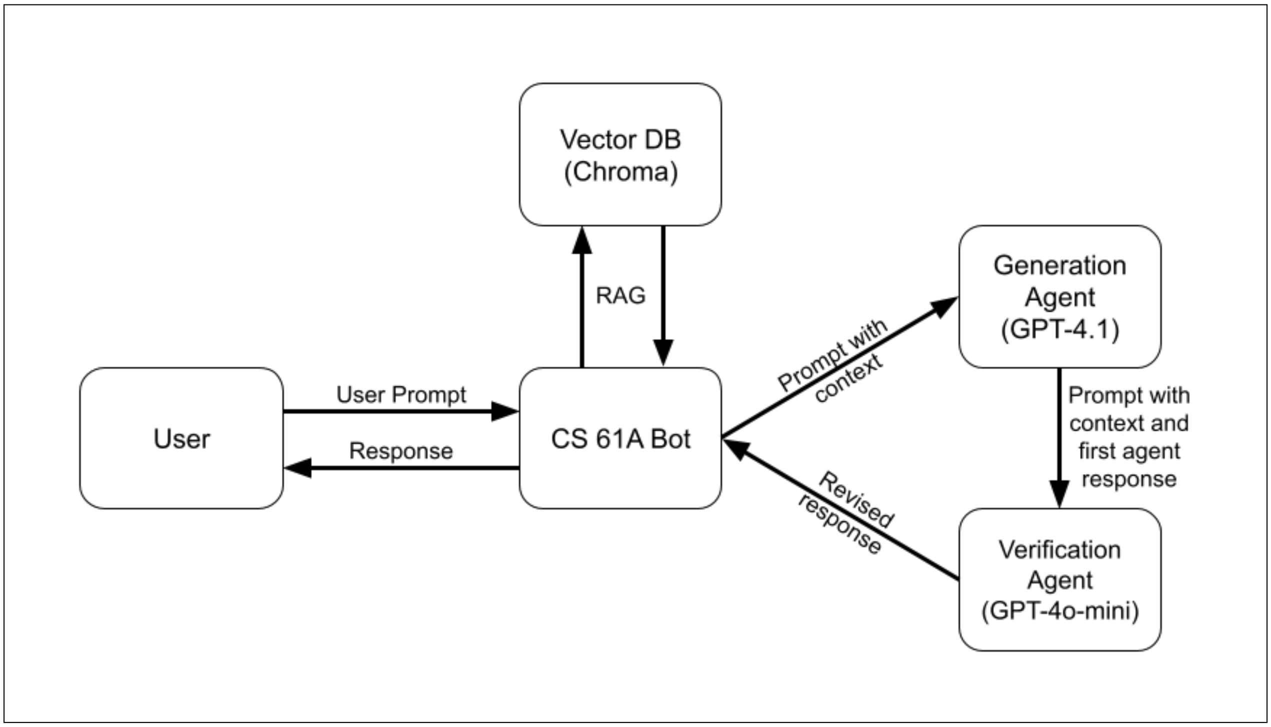


Figure 3.2: The system architecture of the final version of our tutoring bot. The workflow involves two calls to different OpenAI models. In our testing, this architecture produced the most accurate and pedagogically effective results.

3.1 Retrieval-Augmented Generation (RAG)

RAG is a technique known to improve LLM accuracy by grounding responses in external knowledge. When a user asks a question, the system first converts the query into numerical representations called embeddings using an embedding model. It then consults an external database full of relevant information that is split into chunks and also stored as embeddings. The system then attempts to find the most relevant chunks of information based on cosine similarity to the query. This retrieved context is then appended to the user's query and passed to the LLM, guiding it to generate more accurate responses and reducing the chance of hallucination.

For our implementation, we considered several candidates for inclusion in the RAG database, including lecture slides, lecture transcripts, and textbook notes. Ultimately, we selected textbook notes as the notes offered the most organized, structured, and comprehensive source material, qualities that enhance LLM performance. We used OpenAI's text-embedding-3-small for our embedding model, and Chroma for our vector database. For chunking, rather than a naive method of splitting the material by a fixed number of words (e.g., every 100 words), we adopted a more structure-aware strategy by chunking at the level of subsections. This approach leverages the natural organization of the notes into chapters, sections, and subsections (shown in Figure 3.3), and should support more informative retrieval.


```
[
  {
    "type": "h2",
    "content": "1.5 Control"
  },
  {
    "type": "p",
    "content": "The expressive power of the functions that we can define at this point is very limited, because we have not introduced a way to make comparisons and to perform different operations depending on the result of a comparison. Control statements will give us this ability. They are statements that control the flow of a program's execution based on the results of logical comparisons."
  },
  {
    "type": "p",
    "content": "Statements differ fundamentally from the expressions that we have studied so far. They have no value. Instead of computing something, executing a control statement determines what the interpreter should do next."
  },
  {
    "type": "h3",
    "content": "1.5.1 Statements"
  },
  {
    "type": "p",
    "content": "So far, we have primarily considered how to evaluate expressions. However, we have seen three kinds of statements already: assignment, def, and return statements. These lines of Python code are not themselves expressions, although they all contain expressions as components."
  },
  {
    "type": "p",
    "content": "Rather than being evaluated, statements are executed. Each statement describes some change to the interpreter state, and executing a statement applies that change. As we have seen for return and assignment statements, executing statements can involve evaluating subexpressions contained within them."
  },
```


Figure 3.3: CS 61A's textbook, *Composing Programs.* The textbook comes in HTML structure, which we can leverage for our chunking strategy. This helps our RAG retrieve more coherent and complete chunks of information.

Before storing each chunk, we also tag it with its corresponding chapter. That way, when RAG content is used to support an answer, the bot has knowledge of exactly which chapter the content comes from. This has the added benefit of enabling the bot to cite its source so that students can easily locate and review the relevant material, as seen in Figure 3.4.

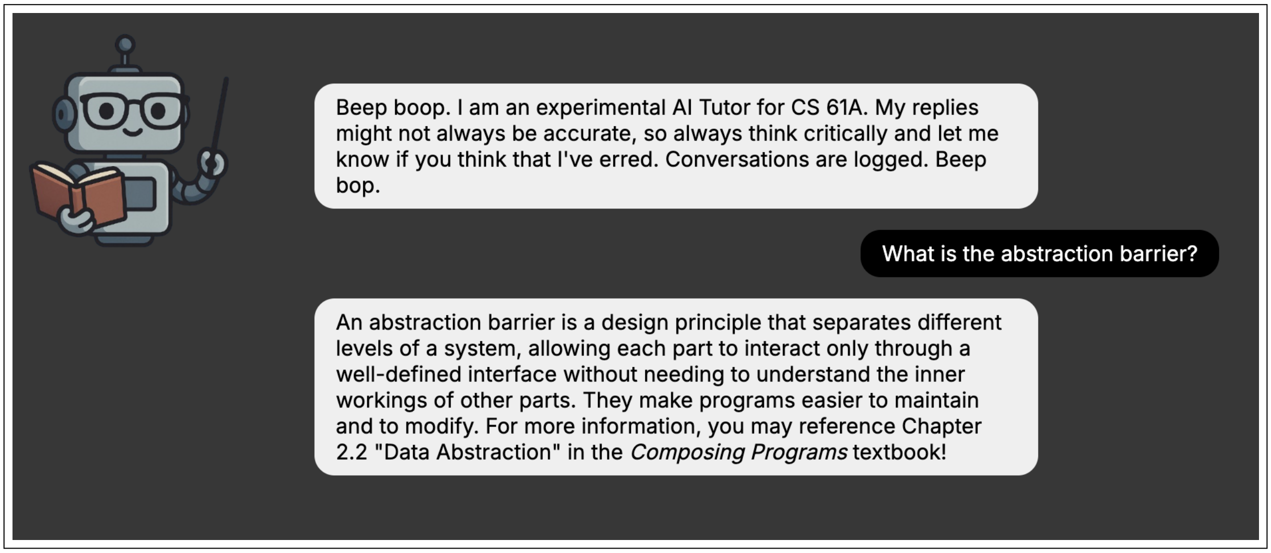


Figure 3.4: Example of the bot citing a source for more information using RAG. Based on their question, the bot suggests the student review Chapter 2.2. This saves the student the trouble of finding the section of the textbook themself, and enables them to easily double-check the bot's answer.

3.2 Dual-Agent System

We also experimented with adding a second LLM agent to the system. Previous LLM tutoring implementations have struggled with two key issues: inaccurate outputs and overly detailed answers that undermine the learning process. Research has shown that introducing a secondary verification agent can dramatically improve overall output quality [13].

In our system, we use GPT-4.1 as the generation agent to create an appropriate response to the user's query. We poll every half-second to check if the first agent is finished generating its output. Once the output is fully generated, we pipe that output into the verification agent (GPT-4o Mini).

The verification agent asks two primary questions: (1) Is the answer correct and consistent with the provided course materials? (2) Does the response avoid giving away the solution? If the previous agent's output satisfies both criteria, the verifier forwards it unchanged; otherwise, the verification agent edits the response before sending it back to the student. Because verification is a simpler task than generation, we found that a smaller model sufficed, which helped to slightly reduce operational costs and improve system latency.

3.3 Prompting

Prompting had one of the largest impacts on the tutor's performance. We experimented with various prompting strategies, informed by OpenAI's guidelines [18]. One of the most consistent challenges we faced in prompt development was balancing the volume of questions the model would ask. While targeted, relevant questions could guide students in the right direction and encourage them to think critically, too many could be annoying to students who were just seeking an answer. Even worse, the model would sometimes answer conceptual questions with its own not-so-relevant question, depicted in Figure 3.7. Though we should not completely answer code-related questions, there is no argument for withholding any information when answering conceptual questions.

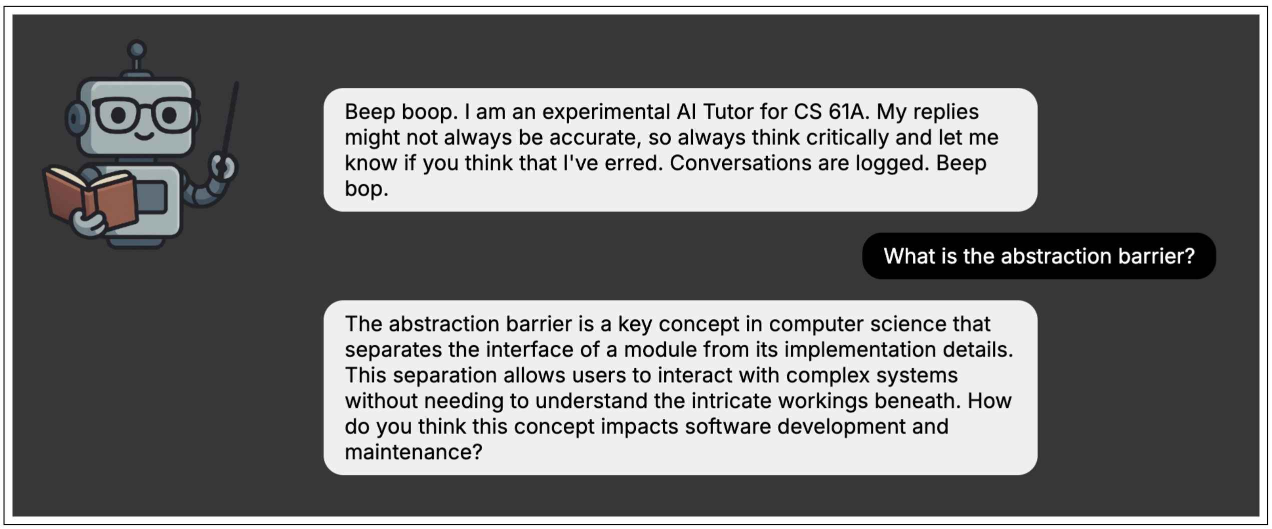


Figure 3.5: Example of an unsuccessful interaction. Early system prompt versions have the bot asking irrelevant questions when responding to conceptual questions. This can be distracting and confusing to a student.

On the other side of the spectrum, if the prompt did not sufficiently emphasize limiting how much help to provide, the model would answer whole problem set questions. It would get around the "no code" restriction by effectively describing the entire code solution in text. In the end, we found that the following prompts provided the best guidance without giving too much away (Figures 3.6 and 3.7).

```
You are CS 61A Bot, a friendly and supportive tutor for CS 61A, the intro CS class at UC Berkeley. Answer student questions only about CS 61A and the field of computer science; do not answer questions about unrelated topics.

Their question, as well as potentially relevant excerpts from the class textbook, Composing Programs, is in the next message. If the excerpt from Composing Programs is helpful, cite it as a source in your answer.

If the student provided code, check if it is correct and complete. If it is, tell them: "That looks like it should work——give it a shot and feel free to include any error message in a comment!" If it is not, here's what you should analyze:

1) Is the code they have already, if any, on the right track? If so, let them know.

2) Is the student missing any conceptual knowledge? Would a refresher help?

3) Do they have a plan——if not, help them generate one.

4) What is the smallest piece of information you think they need in order for them to make progress on their own? Can you ask them a question that leads towards this piece of information?

5) Did they follow your previous instructions? If not, rephrase and offer advice in a different way.


Do not provide full answers to problem sets, as this would violate academic honesty. If there's an obvious bug, direct them to the location of the bug. If there's a conceptual misunderstanding, provide a conceptual refresher, and provide a link to the chapter of Composing Programs in which they can learn more. Limit your response to a few sentences at most. Be as socratic as possible, and be super friendly.

Answer the following question:
```

Figure 3.6: The generation agent's system prompt. The prompt leverages generally accepted strategies of prompt engineering, such as repeating phrases, adding delimiters, and limiting output. This helps guide the bot towards producing a succinct and relevant answer to the student's questions.

```
You are the Verification Agent for CS 61A tutoring.
Your job is to review and edit the previous AI response for accuracy, clarity, and academic appropriateness.

First, check if the previous response contains any code or pseudocode. If so, remove it, and instead provide
  information to help guide the student towards writing this code themselves.
Second, check if the output is consistent with the course materials and learning goals. If the student may
  benefit from reviewing any Composing Programs chapter, make sure the chapter is cited in the response.
Third, ensure that the response is succinct (1 or 2 sentences), socratic, friendly, and free of excess
  technical jargon.

If there are any other issues, edit the response before outputting it.
Always prioritize guiding the student over giving away answers.
If the response is accurate, clear, and does not give away full solutions, output it unchanged.
```

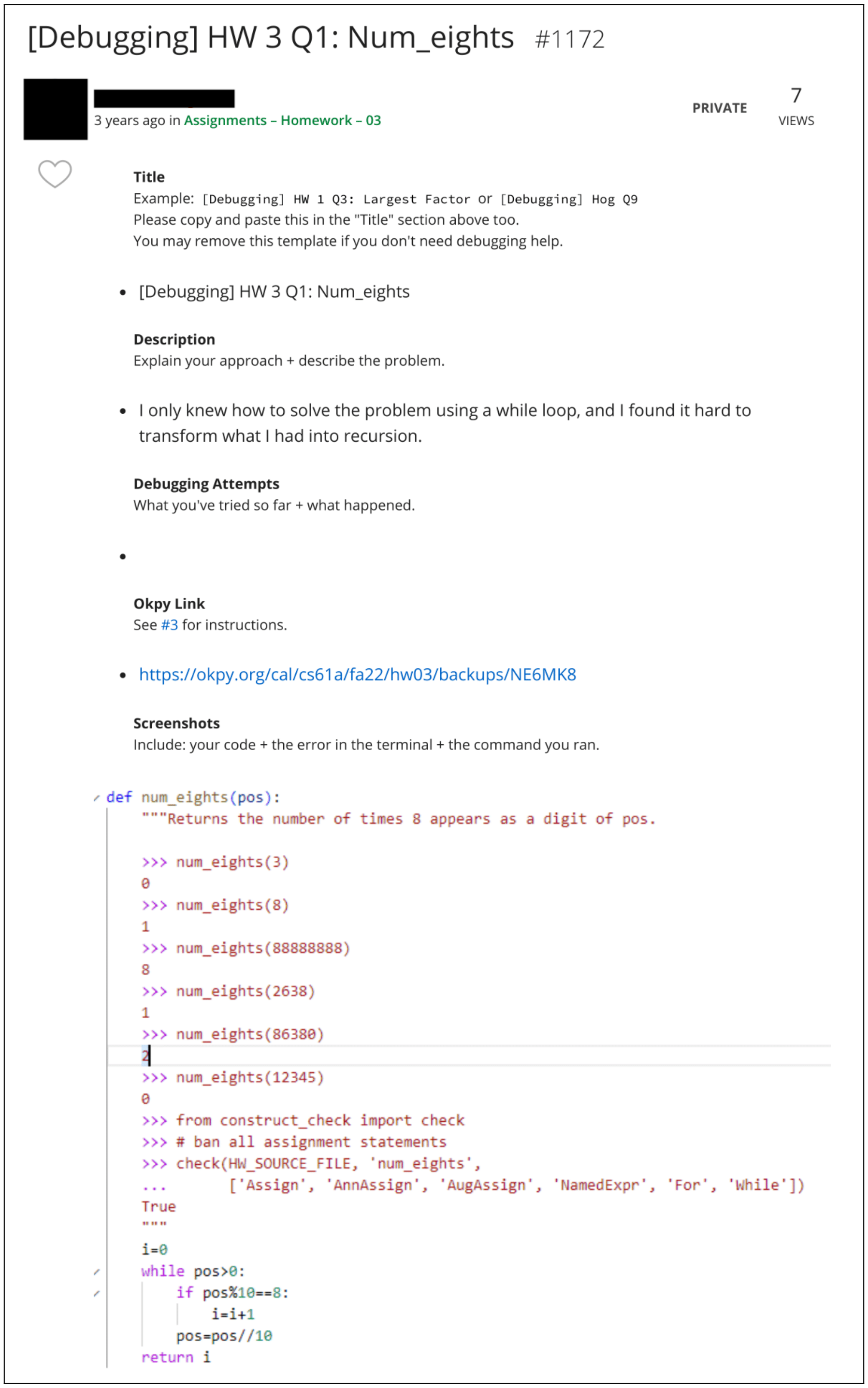
Figure 3.7: The verification agent's system prompt. This prompt checks for the most common mistake by the generation agent, which is over-explaining and providing code. This helps maintain the pedagogical effectiveness of the tutoring system.

In addition to trying to find an optimal prompt wording, we also explored few-shot prompting, in which a few example question–response pairs are included to guide the model's behavior. This technique helps give the model a concrete idea of what tone and style of responses best align with our goals. However, adding too many examples increases prompt length, which inflates both token costs and inference time. Due to the diminishing returns of more examples, we ultimately chose to add two representative question-response pairs to the generation agent's system prompt.

3.4 Fine-tuning

Finally, we applied fine-tuning to further tailor the model to the tutoring use case. Fine-tuning allows small, targeted adjustments to the base model's behavior through additional training on curated examples. We sourced training data from past student–instructor interactions on CS 61A's Ed platform, a centralized forum for class discussions[1]. An example conversation can be seen in Figure 3.8 and Figure 3.9. These interactions closely mirror the modality we wanted for our tutor: text-based student queries (often with code) and detailed instructor responses.

[1] https://edstem.org



[Debugging] HW 3 Q1: Num_eights #1172

[redacted]
3 years ago in Assignments – Homework – 03

PRIVATE

7 VIEWS

Title
Example: `[Debugging] HW 1 Q3: Largest Factor` or `[Debugging] Hog Q9`
Please copy and paste this in the "Title" section above too.
You may remove this template if you don't need debugging help.

- [Debugging] HW 3 Q1: Num_eights

Description
Explain your approach + describe the problem.

- I only knew how to solve the problem using a while loop, and I found it hard to transform what I had into recursion.

Debugging Attempts
What you've tried so far + what happened.

-

Okpy Link
See #3 for instructions.

- https://okpy.org/cal/cs61a/fa22/hw03/backups/NE6MK8

Screenshots
Include: your code + the error in the terminal + the command you ran.

```
def num_eights(pos):
    """Returns the number of times 8 appears as a digit of pos.

    >>> num_eights(3)
    0
    >>> num_eights(8)
    1
    >>> num_eights(88888888)
    8
    >>> num_eights(2638)
    1
    >>> num_eights(86380)
    2
    >>> num_eights(12345)
    0
    >>> from construct_check import check
    >>> # ban all assignment statements
    >>> check(HW_SOURCE_FILE, 'num_eights',
    ...       ['Assign', 'AnnAssign', 'AugAssign', 'NamedExpr', 'For', 'While'])
    True
    """
    i=0
    while pos>0:
        if pos%10==8:
            i=i+1
        pos=pos//10
    return i
```

Figure 3.8: Example question on Ed. Students can ask their questions and provide code, just as they can with the tutoring bot. Questions usually follow a template to make it easier for staff.

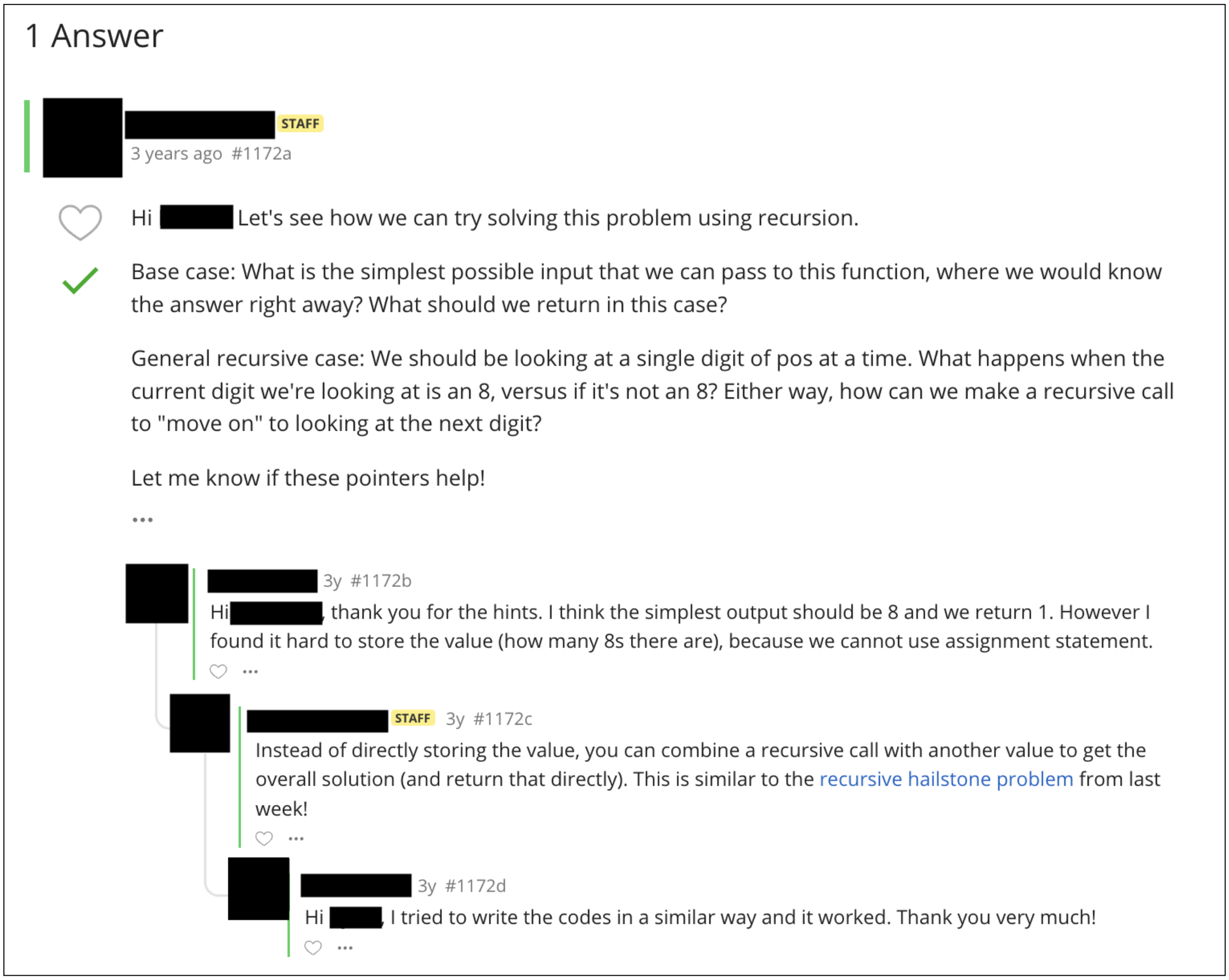


Figure 3.9: Example answer on Ed. A staff member answers in a Socratic manner and without giving away the answer. As a result, the student can think independently, make progress, and then come back if necessary.

In total, we curated a set of 44 high-quality interactions. We mostly copied these interactions from Ed verbatim, but we made minor edits to standardize formatting, correct grammar, and improve conciseness. Figure 3.10 shows the same conversation, converted into a JSON format.

```json
{
  "messages": [
    {
      "role": "system",
      "content": "<Generation Agent System Prompt>"
    },
    {
      "role": "user",
      "content": "I'm struggling with HW3 Q1: num_eights. I only knew how to solve the problem using a while loop, and I found it hard to transform what I had into recursion. Here is my current code:\n\n\```python\ni = 0\nwhile pos > 0:\n    if pos % 10 == 8:\n        i = i + 1\n    pos = pos // 10\nreturn i\n```"
    },
    {
      "role": "assistant",
      "content": "Hi! Let's see how we can try solving this using recursion. Base case: What is the simplest possible input we can pass to the function, where we would know the answer right away? What should we return in that case? General recursive case: We should be looking at one digit at a time. What happens when the current digit is 8, versus when it’s not? Either way, how can we move on to the next digit? Let me know if these pointers help!"
    },
    {
      "role": "user",
      "content": "Thanks for the hints! I think the simplest output should be 8 and we return 1. But I found it hard to store the value (how many 8s there are), because we can't use assignment statements."
    },
    {
      "role": "assistant",
      "content": "Instead of directly storing the value, you can combine a recursive call with another value to get the overall solution (and return that directly)."
    },
  ]
}
```

Figure 3.10: Example JSON conversion of a multi-turn conversation on Ed for fine-tuning. The student and teacher texts are separated and given labels “user” and “assistant”, respectively. This enables the LLM to learn from good tutoring examples and model its responses after the human tutor’s.

This dataset included 12 single-turn conceptual questions, 20 single-turn code-related questions, and 12 multi-turn code-related conversations. These ratios were chosen to roughly mirror the distributions observed in the CS 61A’s Ed platform question types, as well as the frequency of each type of question commonly asked towards LLMs [11]. However, some research suggests that the proportion of multi-turn interactions may increase with the bot compared to human TAs, as students may feel more comfortable asking follow-up questions to an AI tutor [19].

For fine-tuning, we only used 1/4 of the dataset, or 11 samples (3 single-turn conceptual, 5 single-turn code-related, and 12 multi-turn code-related sampled randomly from the total dataset), as we wanted to reserve 3/4 of the dataset for testing.

Chapter 4

Results

To evaluate the impact of each of the techniques on the bots' accuracy and pedagogical soundness, we created three different versions of the tutor bot: V1, a single-agent model with RAG, V2, a dual-agent model with RAG, and V3, a dual-agent model with RAG, few-shot prompting, and fine-tuning. We then tested on the 33 samples distinct from those used in fine-tuning (9 single-turn conceptual, 15 single-turn code-related, and 9 multi-turn code-related questions). Each bot produced responses for all samples, resulting in a total of 99 responses. Then, two human evaluators with teaching experience in CS 61A both independently rated each response for accuracy and pedagogical soundness.

4.1 Accuracy

We defined an accurate response as one that does not provide any false or irrelevant information. Table 4.1 details the evaluators' initial ratings of each version.

Model Version	**Evaluator 1 Accuracy (%)**	**Evaluator 2 Accuracy (%)**
V1	79	82
V2	73	76
V3	76	82

Table 4.1: Evaluation accuracy of model versions V1–V3 by two human evaluators. All of the model responses were shuffled and shown to the evaluators blindly and in random order. This way, we avoid potential bias.

Inter-evaluation agreement measured by Cohen's Kappa was 0.92, indicating almost perfect agreement. The main reasons for discrepancies were differences in strictness for a response to be deemed accurate, and occasional misunderstandings of the student's question.

After resolving these discrepancies through discussion, we obtained the final accuracy scores shown in Table 4.2.

Model Version	Accuracy (%)
V1	79
V2	76
V3	79

Table 4.2: Final agreed upon percentage of accurate responses of versions V1–V3. The worst performing version only provided 1 more inaccurate answer than the rest out of 33 samples. There appears to be no significant difference in accuracy between the models.

4.2 Pedagogical Soundness

We performed a similar process to evaluate the bots' pedagogical soundness. Evaluators judged whether each response met the aforementioned standards for guiding students without directly providing answers. Evaluators only judged pedagogical soundness on responses they deemed accurate. The results are shown in Table 4.3.

Model Version	Evaluator 1 Soundness (%)	Evaluator 2 Soundness (%)
V1	75	79
V2	82	88
V3	88	90

Table 4.3: Evaluation pedagogical soundness of model versions V1–V3 by two human evaluators. As with before, all of the model responses were shuffled and shown to the evaluators blindly and in random order. This way, we avoid potential bias.

Cohen's Kappa for pedagogical assessments was lower, at 0.70, possibly due to the inherent subjectivity in determining "satisfactory" guidance. After resolving discrepancies, we obtained the final ratings shown in Table 4.4.

Model Version	**Pedagogical Soundness (%)**
V1	79
V2	85
V3	90

Table 4.4: Final agreed upon percentage of pedagogically sound responses of model versions V1–V3. Only the samples where both evaluators rated the response as accurate were counted. This was to make sure both evaluators independently came up with a decision for pedagogical soundness before discussing.

It's important to note that because only the samples that were evaluated to be accurate were counted in this statistic, the 90% pedagogical soundness from V3 does not mean that 90% of its responses were indeed pedagogically sound. In reality, only 90% of 79%, or 71% of its responses in the test set, were sound. When also counting for the inaccurate responses, 65% of V2's responses were sound, and only 62% of V1's responses were sound. Unlike with accuracy, however, there appeared to be a significant difference in pedagogical soundness between the models, most notably between V1 and V3.

These numbers reflect an absolute threshold (satisfactory vs. unsatisfactory) but do not necessarily capture relative performance nuances among the versions. Unlike with accuracy, where the information is either correct or incorrect, teaching ability is based on a spectrum. Thus, we also conducted a separate head-to-head evaluation, the results of which can be seen in Figure 4.1. Each evaluator compared the outputs of the three models side-by-side for each data point, ranking them from most to least pedagogically effective. The models' outputs were presented to the evaluators in randomized order, and evaluators were blind to the source of each output in order to avoid bias. Evaluators could assign equal ranks to multiple models if performance was indistinguishable.

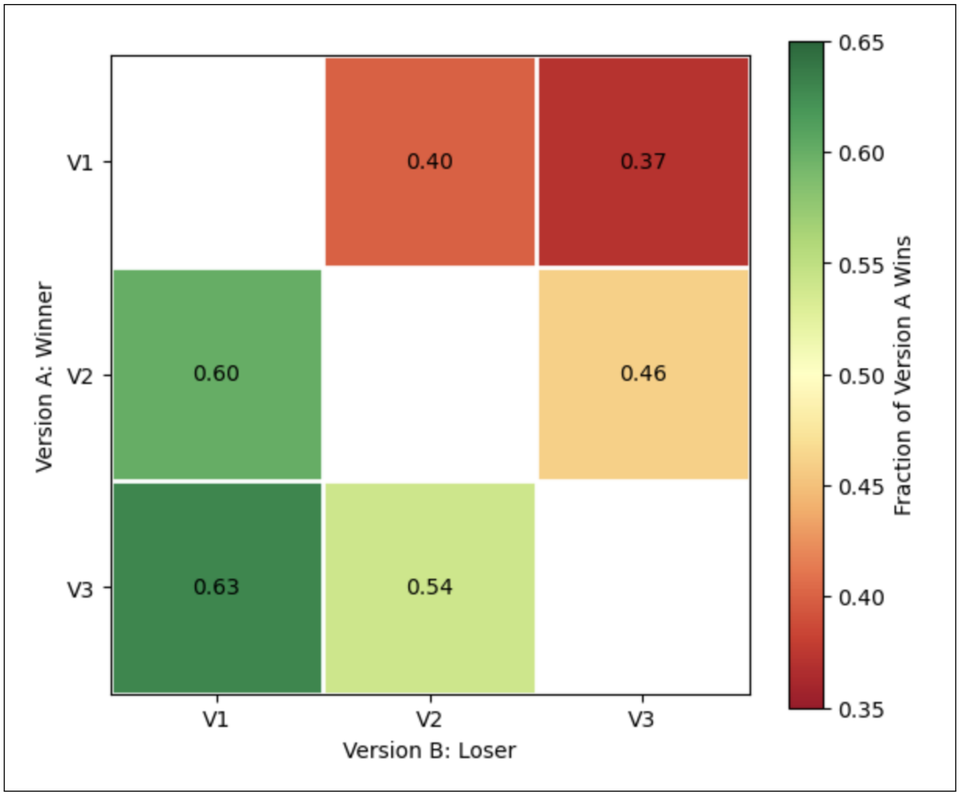


Figure 4.1: Fraction of Model A Wins for All Non-tied A vs. B Battles. Version 3 showed a significant improvement over Version 1, with evaluators preferring its output 63% of the time. This suggests the improvements of Version 3 were effective in generating more pedagogically effective answers.

The results revealed improvements from V1 to V2 and from V2 to V3. Interestingly, the enhanced pedagogical performance observed in V2 and V3 primarily stemmed from improvements in multi-turn interactions. Specifically, the single-agent model (V1) frequently provided overly detailed solutions upon repeated prompts, diminishing its effectiveness in multi-turn contexts. This is possibly due to context drift and an LLM's inherently limited attention capacity. In long interactions, earlier instructions get pushed further back in the input sequence, reducing their influence relative to recent dialogue. The presence of the verification agent in V2 and V3, along with the further enhancements through prompting and fine-tuning, seemingly helped mitigate this issue by acting as a persistent second-stage filter that enforced consistency with the system's instructional goals.

Notably, these improvements in pedagogy were not mirrored in the accuracy ratings. This may indicate that the inaccuracies from the generation model were largely undetectable by the verification model.

4.3 Latency

An important consideration in evaluating these models is the time each one takes to produce a response. Figure 4.2 shows the response times of each model on the 33-sample test set.

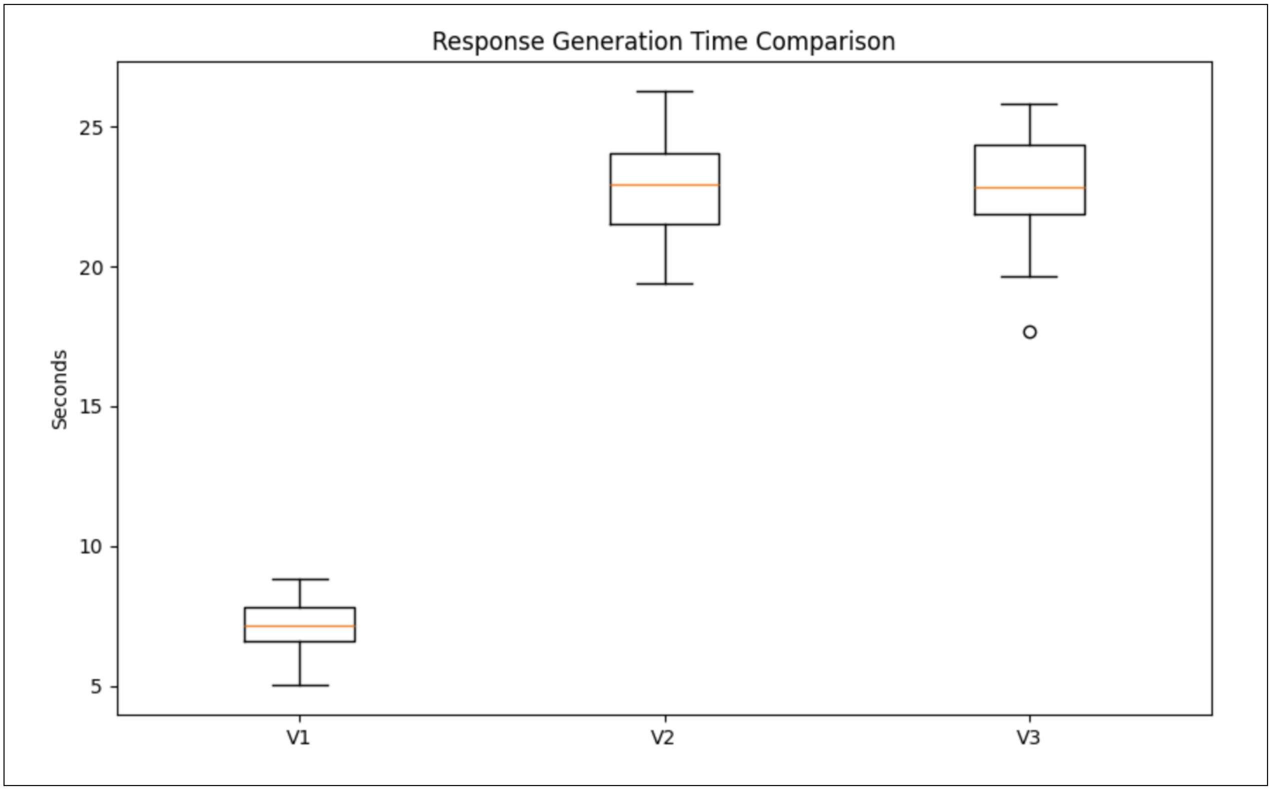


Figure 4.2: Response times of each version across 33 samples. There is a significant difference between V1 and the rest at a 95% confidence interval. There appears to be no measurable difference between V2 and V3, however.

On average, V1 responded in approximately 7 seconds, while V2 and V3 each took around 23 and 22 seconds, respectively. The significant increase in response time from V1 to V2/V3 appears to be due to the introduction of a second verification agent. In our implementation, the verification agent only began processing after the generation agent had completed its entire output, with polling occurring every 0.5 seconds to check for completion. This sequential structure added a substantial delay. In future iterations, we want to explore streaming the generation agent's output to the verification agent, a technique that has been shown to significantly reduce delay [3].

Interestingly, there was little difference in latency between V2 and V3, suggesting that adding two additional few-shot examples to the generation agent's system prompt had minimal impact on response speed.

The effect of this latency on user experience remains uncertain. Drawing from studies on user behavior with web page load times, delays of over 20 seconds may deter users from engaging with the system altogether [2]. On the other hand, in a resource-constrained environment where there is no other teacher, a motivated student may be willing to accept the slow response time. Furthermore, a slow response time could serve as an implicit rate limiter, restricting use and encouraging students to spend time reflecting before submitting further questions.

Chapter 5

Discussion

5.1 Positive vs. Negative Student-Tutor Bot Interactions

Beyond the quantitative metrics, a closer examination of individual student-bot exchanges revealed some broader patterns. Overall, the bot demonstrated strong performance on conceptual questions. This is likely due to the rich set of notes fed into the RAG pipeline, along with the absence of strict constraints on how much assistance the bot could provide in these cases.

For problem-solving questions, successful interactions, like the one shown in Figure 5.1, typically involve the bot offering targeted guidance through well-posed questions, succinct explanations, and constructive hints. In the absence of a human instructor, version 3 of the bot may be able to serve as a reasonable stand-in, especially in resource-limited situations.

The most challenging interactions tended to occur in extended, multi-turn conversations. In particular, when a student already had a specific solution path in mind, the bot would sometimes diverge from that path, proposing a different, albeit technically correct, method. For example, finding the nth number of the Fibonacci sequence is a classic exercise for students learning recursion. During testing, we found that the bot gave helpful advice when students were lost or were working through a standard solution. However, if you fed it a unique, suboptimal solution, such as keeping a list of previous numbers in the sequence through each recursive call, the bot would struggle to advise students. Usually, it would ignore the student's progress and start hinting towards the recommended solution. If forced to continue the student's solution with repeated prompting, the bot would frequently advise incorrect changes. While this is a common pitfall even for novice human tutors, it's an area for improvement in future versions of the bot.

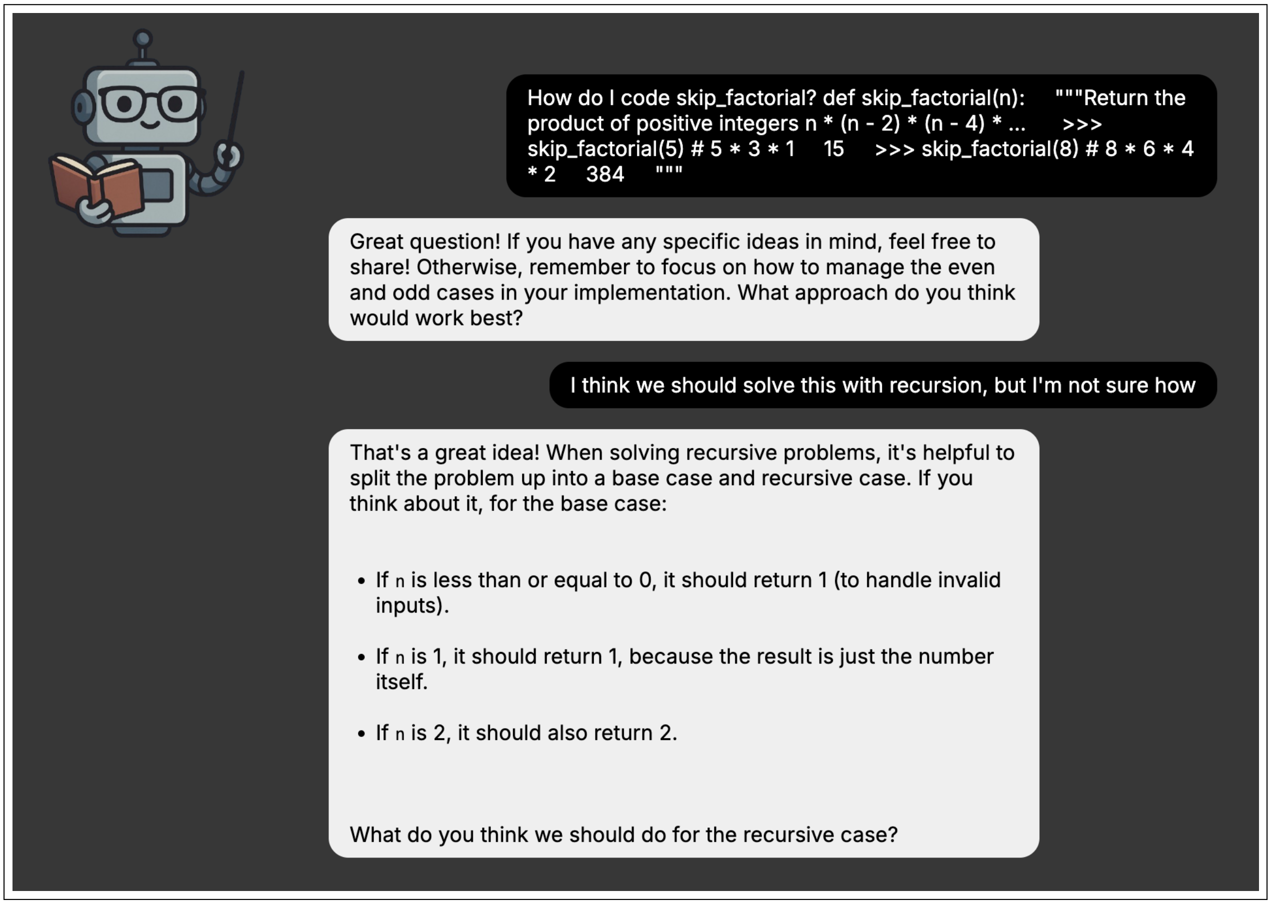


Figure 5.1: Example of a positive interaction. The bot gives positive affirmation to the student's idea, provides hints without code, and asks the student a question about what to do next. This bears similarities to what a good human tutor would do.

5.2 Applicability to Other Courses

Looking forward, we envision that tools like this could support learning in disciplines beyond computer science. To test this, we developed a prototype bot for UC Berkeley's UGBA 102B Managerial Accounting, using the same architecture. We ingested the course slides into the vector database and adjusted the prompts to reflect the new subject matter.

This version, effectively a V2 bot due to the lack of few-shot prompting or fine-tuning, was tested on a few basic queries. The results were mixed. While the bot was able to handle conceptual questions and interpret student intent well (see Figure 5.2), it struggled with problems that required multi-step reasoning or calculations.

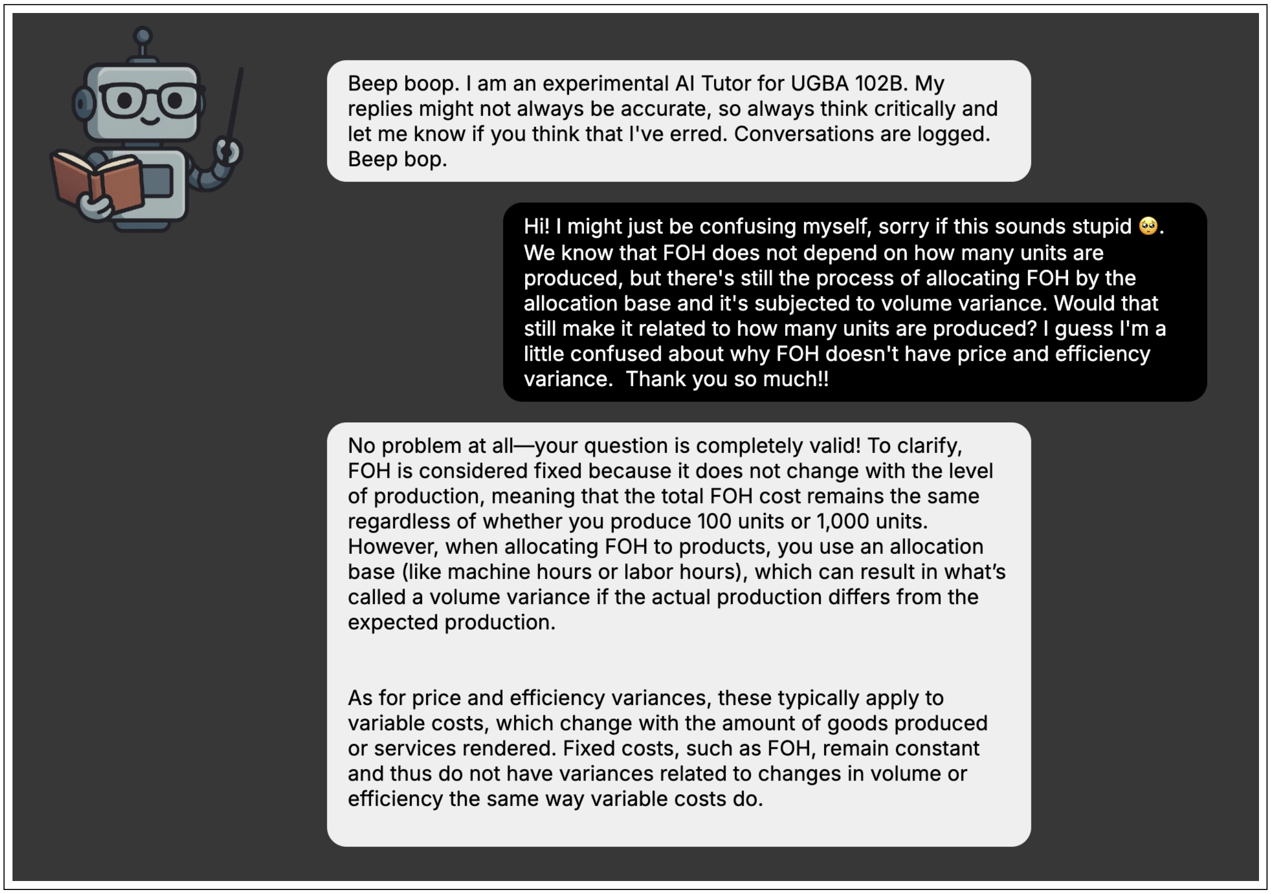


Figure 5.2: Example conversation from a bot designed for an introductory accounting course. Based on the class notes and system prompting, it can interpret what "FOH" means in this context. It answers the student's questions with an example and maintains a friendly tone, facilitating student learning.

We suspect this decline in problem-solving performance stems from two factors. First, the accounting course materials lacked structured, comprehensive notes, making it harder for the LLM to extract relevant information. Second, foundational models themselves likely have less available accounting content in their training data compared to computer science, reducing their baseline competence in the domain.

Chapter 6

Limitations & Future Work

While initial evaluations conducted with teaching assistants indicate promising results, we have yet to conduct any testing directly with students. Future research should involve formal student evaluations to better understand how learners interact with the tool, as well as to systematically identify its strengths, weaknesses, and areas for improvement. Collaborating with an academic course, for instance, could provide a robust dataset to evaluate the tool's effectiveness, usability, and impact on student learning outcomes. Conducting such a study would require obtaining Institutional Review Board (IRB) approval and finding a willing and appropriate class to partner with. Partnering with a course and university could have the additional benefit of potential for longer-term studies that could examine the sustained effects of AI tutoring on student performance and retention of material, which is critical to understanding the lasting educational impacts of these tools.

Also, to broaden accessibility and adoption, we plan to simplify the deployment process. Currently, the codebase is publicly available via GitHub[1], and anyone can clone it, acquire an OpenAI API key, and tailor it for their specific course or use case. However, doing so would require some baseline degree of computer science background and understanding. To address this, we aim to develop a hosted web interface that simplifies the setup. We envision a basic form-like page that allows people to submit their course or topic of interest and upload any course materials. Upon submitting all of the necessary information, they will have access to a simplified version of the tutoring bot for their own use case. We will provide options for data collection—allowing users to choose between anonymized data tracking or complete privacy—which would encourage broader adoption while respecting user preferences.

Moreover, future research should investigate the effectiveness of this framework for subjects beyond introductory computer science. While we believe the tool could be adapted for other domains, and see some promising data with introductory accounting, effectiveness might vary, particularly in fields that emphasize creativity or subjective interpretation rather than structured problem-solving. Earlier findings highlighted instances where the bot struggled with building on top of creative, unconventional student approaches, underscoring the

[1]https://github.com/mykolwu/llm-tutoring-bot

need to explore these limitations further.

Lastly, more comprehensive studies are needed to explore the potential social and psychological impacts associated with substituting human interaction with AI support. Would 24/7 access to a tutor discourage the necessary independent work and struggle needed for effective learning? Furthermore, past research indicates that women in computer science programs are more likely to utilize office hours and LLM resources [9, 11]. This is possibly due to there being fewer women in the field, limiting access to peer networks. If this holds true, widespread adoption of accessible, AI-based educational resources could help promote equity within computer science education. Further targeted investigation into these dynamics is necessary to fully understand.

Chapter 7

Conclusion

In this work, we have demonstrated that carefully designed LLM-based tutoring bots can provide relatively accurate and pedagogically effective help for introductory computer science. We found that employing a dual-agent validation system, structured retrieval-augmented generation (RAG), and fine-tuning with targeted few-shot prompting can improve a model's ability to guide students effectively. Specifically, we found that the additional validation agent was especially effective in prolonged, multi-turn conversations, albeit at the cost of a slower and more expensive querying process. Nonetheless, the significant improvement in output suggests that future attempts towards creating an LLM-based tutoring bot should strongly consider using multiple agents, as well as the standard techniques of RAG, fine-tuning, and few-shot prompting.

However, as LLM technology continues to advance, we anticipate that costs and inference time will decrease, while the effectiveness and reliability of AI tutors will increase, making personalized tutoring more accessible and scalable than ever before. LLMs represent a promising solution for addressing educational inequities by providing high-quality, personalized feedback and instruction to students regardless of their geographic or socioeconomic status. At schools and universities, it may also help alleviate the burden of overloaded teaching staff. However, it is critical to emphasize that these benefits can only be fully realized if LLMs are responsibly used as learning tools rather than shortcuts to bypass the learning process.

Ultimately, we envision a future in which quality education becomes universally accessible through the thoughtful integration of AI technologies. With these findings and methodologies, we aim to contribute to the growing body of best practices that can guide us to that future.

Bibliography

[1] Yongjun An, J.H. Yu, and S. James. “Investigating the higher education institutions’ guidelines and policies regarding the use of generative AI in teaching, learning, research, and administration”. In: *International Journal of Educational Technology in Higher Education* 22.10 (2025). DOI: 10.1186/s41239-025-00507-3. URL: https://doi.org/10.1186/s41239-025-00507-3.

[2] Fiona Fui-Hoon Nah and. “A study on tolerable waiting time: how long are Web users willing to wait?” In: *Behaviour & Information Technology* 23.3 (2004), pp. 153–163. DOI: 10.1080/01449290410001669914. eprint: https://doi.org/10.1080/01449290410001669914. URL: https://doi.org/10.1080/01449290410001669914.

[3] Anonymous. “Staircase Streaming for Low-Latency Multi-Agent Inference”. In: *Submitted to ACL Rolling Review - August 2024*. under review. 2024. URL: https://openreview.net/forum?id=4DVWmeHLNa.

[4] Brett A. Becker et al. “Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, pp. 500–506. ISBN: 9781450394314. DOI: 10.1145/3545945.3569759. URL: https://doi.org/10.1145/3545945.3569759.

[5] Benjamin S. Bloom. “The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring”. In: *Educational Researcher* 13.6 (1984), pp. 4–16. DOI: 10.3102/0013189X013006004. URL: https://journals.sagepub.com/doi/10.3102/0013189X013006004.

[6] Paul Denny, Viraj Kumar, and Nasser Giacaman. *Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language*. 2022. arXiv: 2210.15157 [cs.HC]. URL: https://arxiv.org/abs/2210.15157.

[7] Irene Estévez-Ayres, Pablo Callejo, María Ángeles Hombrados-Herrera, et al. “Evaluation of LLM Tools for Feedback Generation in a Course on Concurrent Programming”. In: *International Journal of Artificial Intelligence in Education* (2024). Accepted: 13 April 2024, Published: 15 May 2024. DOI: 10.1007/s40593-024-00406-0. URL: https://doi.org/10.1007/s40593-024-00406-0.

[8] James Finnie-Ansley et al. "The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming". In: *Proceedings of the 24th Australasian Computing Education Conference*. ACE '22. Virtual Event, Australia: Association for Computing Machinery, 2022, pp. 10–19. ISBN: 9781450396431. DOI: 10.1145/3511861.3511863. URL: https://doi.org/10.1145/3511861.3511863.

[9] Zhikai Gao, Sarah Heckman, and Collin Lynch. "Who Uses Office Hours?: A Comparison of In-Person and Virtual Office Hours Utilization". In: Feb. 2022, pp. 300–306. DOI: 10.1145/3478431.3499334.

[10] Onur Karademir et al. "I don't have time! But keep me in the loop: Co-designing requirements for a learning analytics cockpit with teachers". In: *Journal of Computer Assisted Learning* 40.6 (2024), pp. 2681–2699. DOI: https://doi.org/10.1111/jcal.12997. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jcal.12997. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/jcal.12997.

[11] Majeed Kazemitabaar et al. "CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. CHI '24. ACM, May 2024, pp. 1–20. DOI: 10.1145/3613904.3642773. URL: http://dx.doi.org/10.1145/3613904.3642773.

[12] Juho Leinonen et al. "Comparing Code Explanations Created by Students and Large Language Models". In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2023. Turku, Finland: Association for Computing Machinery, 2023, pp. 124–130. ISBN: 9798400701382. DOI: 10.1145/3587102.3588785. URL: https://doi.org/10.1145/3587102.3588785.

[13] Hunter Lightman et al. *Let's Verify Step by Step*. 2023. arXiv: 2305.20050 [cs.LG]. URL: https://arxiv.org/abs/2305.20050.

[14] Rongxin Liu et al. "Improving AI in CS50: Leveraging Human Feedback for Better Learning". In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSETS 2025. Pittsburgh, PA, USA: Association for Computing Machinery, 2025, pp. 715–721. ISBN: 9798400705311. DOI: 10.1145/3641554.3701945. URL: https://doi.org/10.1145/3641554.3701945.

[15] Rongxin Liu et al. "Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2024. Portland, OR, USA: Association for Computing Machinery, 2024, pp. 750–756. ISBN: 9798400704239. DOI: 10.1145/3626252.3630938. URL: https://doi.org/10.1145/3626252.3630938.

[16] National Academies of Sciences, Engineering, and Medicine. *Assessing and Responding to the Growth of Computer Science Undergraduate Enrollments*. Washington, DC: The National Academies Press, 2017. DOI: 10.17226/24926. URL: https://doi.org/10.17226/24926.

[17] Sydney Nguyen et al. "How Beginning Programmers and Code LLMs (Mis)read Each Other". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. CHI '24. ACM, May 2024, pp. 1–26. DOI: 10.1145/3613904.3642706. URL: http://dx.doi.org/10.1145/3613904.3642706.

[18] OpenAI. *OpenAI Platform Documentation: Text Completion Guide.* https://platform.openai.com/docs/guides/text?api-mode=responses. Accessed: 2025-04-30. 2025.

[19] Laryn Qi. "Developing and Evaluating LLM Assistants in Introductory Computer Science". MA thesis. EECS Department, University of California, Berkeley, May 2024. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-132.html.

[20] Laryn Qi et al. *A Knowledge-Component-Based Methodology for Evaluating AI Assistants*. 2024. arXiv: 2406.05603 [cs.CY]. URL: https://arxiv.org/abs/2406.05603.

[21] Xiaohang Tang et al. *SPHERE: Scaling Personalized Feedback in Programming Classrooms with Structured Review of LLM Outputs*. 2024. arXiv: 2410.16513 [cs.HC]. URL: https://arxiv.org/abs/2410.16513.

[22] Rose E. Wang et al. *Tutor CoPilot: A Human-AI Approach for Scaling Real-Time Expertise*. 2025. arXiv: 2410.03017 [cs.CL]. URL: https://arxiv.org/abs/2410.03017.

[23] Hiroko Kawaguchi Warshauer. "Productive Struggle in Middle School Mathematics Classrooms". In: *Journal of Mathematics Teacher Education* 18.4 (2015), pp. 375–400. DOI: 10.1007/s10857-014-9286-3. URL: https://doi.org/10.1007/s10857-014-9286-3.

[24] Jeremy Weissman. "ChatGPT Is a Plague Upon Education". In: *Inside Higher Ed* (Feb. 2023). URL: https://www.insidehighered.com/views/2023/02/09/chatgpt-plague-upon-education-opinion.

[25] Mike Wu et al. *ProtoTransformer: A Meta-Learning Approach to Providing Student Feedback*. 2021. arXiv: 2107.14035 [cs.CY]. URL: https://arxiv.org/abs/2107.14035.