Promoting Learning over Completion via Prompt Engineering within the LeafTutor Agent

Timothy Carter*, Ariful Islam Emon*, Ryan Jackson*, Pu Tian†, and Yalong Wu*
*Department of Computing Sciences, University of Houston–Clear Lake, TX, USA
Emails: {cartert2306, emona3832, jacksonr2411, wuy}@uhcl.edu
†School of Business, Stockton University, NJ, USA. Email: pu.tian@stockton.edu

***Abstract*—Recent advances in large language models (LLMs) enable new approaches to programming education, but direct use often prioritizes task completion over conceptual understanding. This paper introduces LeafTutor, an AI tutoring agent that guides students through programming assignments using structured prompting and controlled interaction. The system constrains solution disclosure while promoting learning through principles inspired by sports psychology—such as goal decomposition and progress reinforcement—and a "chat-free" interface that reduces interaction friction through predefined guidance options. We evaluate the system using metrics including accuracy, depth, noise, efficiency, flow, redundancy, and completion to compare LLM performance within this framework. Results show that constraining LLM behavior improves both response quality and educational effectiveness. This work provides a structured approach for aligning LLM-based tutoring systems with pedagogical objectives.**

***Keywords*—AI agent, Tutor, LLM, Prompt Engineering**

I. Introduction

Recent advances in large language models (LLMs) such as ChatGPT and Claude have significantly expanded the capabilities of AI systems in natural language understanding and generation, enabling new applications in programming education and intelligent tutoring [1], [2]. These models are capable of generating accurate, context-aware responses across a wide range of domains, making them attractive tools for assisting students in learning complex technical concepts. However, despite their strengths, LLMs are not inherently designed to support structured learning. Their default behavior prioritizes task completion, often producing fully formed solutions rather than guiding users through the reasoning process required to develop conceptual understanding [3], [4].

This limitation poses a significant challenge in educational contexts, particularly in programming, where learning is iterative and concept-driven. Effective tutoring requires more than correct answers; it requires guiding students through problem decomposition, reinforcing intermediate concepts, and encouraging active engagement. This aligns with prior work in intelligent tutoring systems, which emphasizes structured guidance and incremental learning as key drivers of student success [5]. Direct interaction with LLMs frequently bypasses these critical stages, leading to shallow understanding and reduced knowledge retention. As a result, while LLMs are powerful tools for answering questions, they are not sufficient on their own to function as effective tutors.

A key distinction underlying this limitation is the difference between standalone LLMs and AI agents. LLMs operate as reactive systems, generating outputs based solely on input prompts without maintaining control over interaction flow or enforcing pedagogical structure. In contrast, AI agents incorporate additional layers of logic, including contextual awareness, behavioral constraints, and response regulation, enabling them to guide interactions toward specific objectives [6], [7]. In educational settings, this distinction is crucial, as it allows for the transformation of LLMs from passive response generators into systems capable of supporting structured learning experiences.

The need for such structured guidance is particularly evident in programming education, where students must develop both conceptual understanding and problem-solving skills. Research in learning science highlights that active engagement and retrieval-based practice significantly improve long-term knowledge retention [8]. Without structured interaction, students may become overly reliant on generated solutions, undermining the learning process.

To address these challenges, we introduce *LeafTutor*, an AI tutoring agent designed to promote conceptual understanding through controlled interaction. The system constrains direct solution disclosure while guiding students using structured prompting strategies and agent-based control mechanisms [9]. In addition to these technical components, LeafTutor incorporates principles from sports psychology, such as goal decomposition, cognitive stabilization, and progress reinforcement in order to improve engagement, reduce anxiety, and support sustained learning [10]. These principles help structure the learning experience in a way that mirrors effective human coaching strategies.

A distinguishing feature of LeafTutor is its "chat-free" interaction design, which provides users with predefined guidance options rather than relying solely on open-ended input. This approach reduces cognitive load, minimizes interaction friction, and ensures more consistent guidance throughout the learning process. By limiting unstructured interactions, the system maintains tighter control over response quality and instructional progression.

To evaluate the effectiveness of this approach, we define a comprehensive set of performance metrics, including accuracy, depth, noise, efficiency, flow, redundancy, and completion. These metrics are used to compare LLM performance within

the constrained tutoring framework, providing insight into how agent-based design influences both response quality and educational value.

The contributions of this work are threefold. First, we present a structured agent-based framework for transforming LLMs into effective tutoring systems. Second, we introduce an evaluation methodology that captures both quantitative performance and instructional quality. Third, we demonstrate that by combining prompt engineering, agent control, and pedagogically informed design, we can significantly improve the effectiveness of AI-assisted programming education.

Overall, this work highlights the importance of integrating system-level constraints and educational principles when applying LLMs to learning environments, and provides a foundation for future research in AI-driven tutoring systems.

The remainder of this work is organized as follows. In Section II, we cover related works. Section III describes the LeafTutor agent, Section IV outlines our prompt design. In Section V, we introduce metrics, conduct an experiment with our LeafTutor agent, and discuss results. Section VI provides suggestions for future directions, and we conclude in Section VII.

II. Related Works

Previous attempts have been made at Intelligent Tutoring Systems (ITS) architectures with the goal of using advances in technology to assist students in learning [5]. However, LLMs within the ITS architecture are still developing [7]. More specifically, ITS has been applied to assisting students in the Computer Science field developing [11]–[14]. Promising aspects of previous research shows there are benefits that can mimic human tutors [13], tutor agents are capable of providing student-specific assistance [14], feed-back can be student-specific [12], learning speed can be accelerated [11], and, importantly, motivational support can be provided [12]. Despite these positive advances, measuring effectiveness beyond qualitative categorization limits the ability to place one technique or tool over another approach or system [13]. Previous attempts have also noted shortcomings in providing solutions too soon [14] and over-reliance on the tool, where a student may complete the assignment but may have not 'learned' the concept [11], [12].

While existing research regarding tutor agents tends to emphasize conceptualization and description at the architectural level, in this paper we shift focus on driving tutor agent behavior very specifically through internal prompt engineering. Simply put, this internal prompt is part of the constraining nervous system of the agent and is combined along with other elements within the agent such as the student question before the question is passed to the LLM for reasoning. (See Figure 2.) Within this ecosystem, the internal prompt guides LLM decision-making, and alters the response passed back to the student, tailoring responses beyond standalone student-LLM interaction.

Since metrics within the domain of tutor agent systems are not well established, we must rely upon prompt engineering theory when framing the internal prompt in order to rigorously ground our approach. Prompt engineering provides guidelines for the questions passed to the LLM with the goal of improving the subsequent response specific to the task at hand [9]. Examples of prompt engineering used within this paper include expressly defining the role of the agent within the prompt, (e.g. telling the agent "You are a tutor"), defining the task (e.g. "Break the assignment into smaller elements."), and setting behavior limits (e.g. "Do not..."), among others described in detail within Section IV.

With prompt engineering as the framework to deliver our tutoring fundamentals in a manner that improves LLM responses to the student, we lean upon additional research to fill prompt content with evidence-based learning principles and supportive psychology. These learning principles establish that hurried completion of a given task does equate to actual learning, and ingraining concepts requires a tolerance for the uncomfortable place between not-knowing and knowing [8], [15]. (See Figure 3.) To provide emotional support to raise this tolerance, we borrow from sports psychology [10] which helps high-performance athletes mentally prepare to achieve hard things.

Combining prior works noting the strengths and weaknesses of previous ITS, prompt engineering theory, established learning principles and sports psychology, we seek to focus

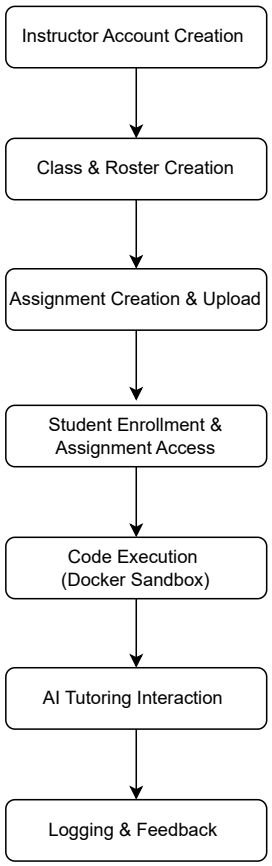

Fig. 1. Overall workflow of the *LeafTutor* system, from instructor account creation and material ingestion to student assignment interaction and AI-assisted feedback.

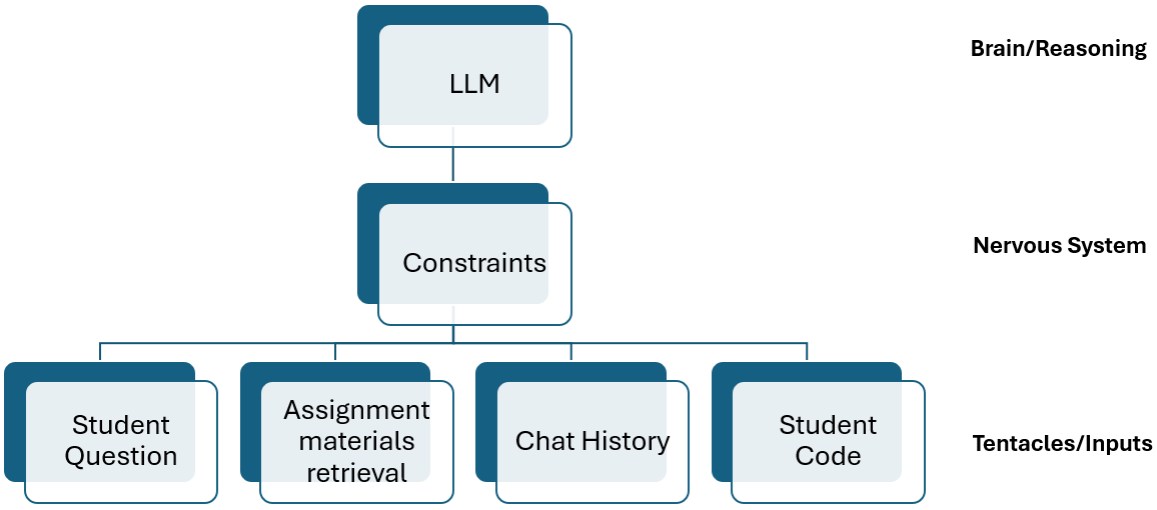

Fig. 2. Conceptual agent framework is similar to an octopus with the tentacles, brain, and nervous system mirroring the inputs, LLM, and constraints such as the internal prompt, respectively.

attention on structuring the agent in such a manner that learning is emphasized over task completion directed via "chat-free" guidance options that encourage student exploration of the topic to incite curiosity and critical thinking, by promoting student persistency through encouragement over one-time 'correctness' as well as limiting final solutions in an effective manner.

III. Leaf Tutor

The purpose of this section is to provide a brief conceptual overview of how our tutor agent functions. First, we will discuss the flow of the assignment, another advantage of our tutor agent over direct LLM interaction by the student.

As demonstrated in Figure 1, the instructor creates an account within the agent and sets up the class and roster of students. He or she then uploads assignments directly applied to that particular class and group of students. Thus, the agent now has knowledge of the assignment and is better prepared to provide direction before the student begins. Next, the student logs in and can begin asking questions right away. The session is then logged and feedback provided.

As a mental model, one may consider the conceptual framework of this agent to be octopus. This octopus has three key parts: tentacles, brain, and nervous system. See Figure 2

Its tentacles are the various input receptors for the agent. These include the student's question, assignment materials retrieved from the professor, the chat history between the student and the agent for context, and any student code that has been provided.

The brain in this mental model is the LLM with reasoning capabilities. It will process the inputs described above and provide a response, in this case that would be similar to the octopus actually moving.

The nervous system's role in this conceptual framework is a control mechanism that sits along the path between inputs and the brain. It constrains inputs in such a manner to shape the agent's behavior. It can be thought of as the nervous system amplifying or downplaying sensory signals passed to the brain. The nervous system for this agent includes four main categories.

First is context selection where it is decided what to include in the LLM request by getting, ranking, and redacting text chunks called "snippits." Next, policy controls what the tutor is allowed to show and when. For example, it may redact portions of the professor's assignment that includes the answer. Another part of this nervous system analogy is state progression that tracks session state over time and changes tutor behavior as the student interacts. The agent will behave differently in later interactions than at the beginning. And finally, the internal agent prompt decides how the LLM is instructed and how selected context is framed. It is this latter portion of the agent's nervous system that is the focus of this paper, namely prompt engineering to optimize the agent.

Comparatively, direct student-LLM interaction would remove tentacles and the nervous system within this analogy, leaving the student to wrestle directly with the LLM in a manner less tailored towards the specific task.

IV. Prompt Design

In this section we structure our internal Leaf Tutor prompt that constrains agent behavior by briefly establishing what good tutor behavior looks like, the concepts our prompt must entail to mimic this behavior, and building these concepts within prompt engineering theory frameworks. The combined prompt can be broken into three portions (P1-P3), representing encouragement, prerelease prevention, and chat-free guidance, respectively. The combination of each portion to create the final prompt can be visualized in Figure 4.

As we design prompts to tailor the agent to perform as a tutor, we must align them with the overall objective of a tutor - helping students learn concepts as they progress toward assignment completion. To visualize this approach consider the following graphic Figure 3, where a student is on one end and

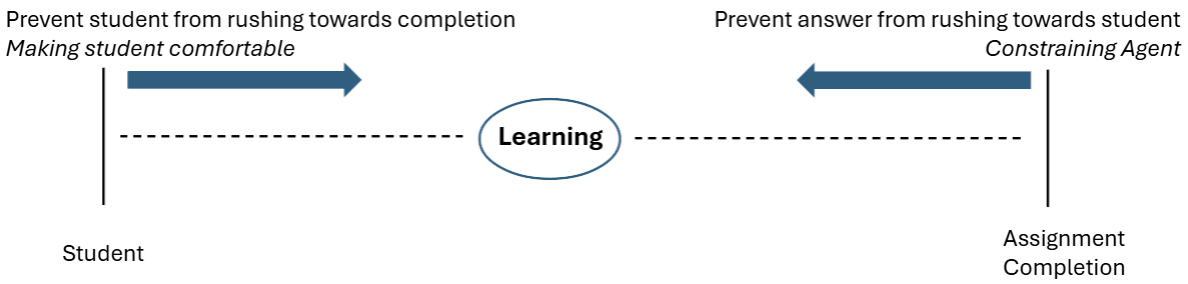

Fig. 3. To maximize learning v. completion, the agent must not provide the answer too soon and must dissuade the student from rushing to the answer.

assignment completion is on the other end. Our objective as a tutor is not to rush the student towards assignment completion. The more time a student spends comfortably guided in the journey between not-knowing and knowing, the better they engrain and "learn" concepts, as opposed to memorization or simple completion [8], [15]. The essence is captured by the common phrase, "Confusion is the path to enlightenment."

Thus, our goal related to our Tutor is prevent the rush to completion in two ways. One, reduce the student's perceived need to rush towards the answer by providing an encouraging environment (P1). Two, we prevent the answers from moving too rapidly towards the student by constraining the AI Tutor to not prematurely release the answer (P2). With the answer constrained and the student relaxed, the time spent in the "learning" phase is maximized. Finally, to ensure the time within "learning" phase is well-spent, we progress the student along with directed, easy chat-free guidance options (P3).

This approach addresses previously recognized roadblocks to tutors within prior attempts. It has been noted that student learning with AI assistance can remain a challenge [11], [12]. The release of the answer must be constrained against pre-release [11], [12], [14]. Encouragement is an integral part of AI-guided learning [5], [12], and that it is problematic to establish benchmarks placing one AI system over another [7], [13].

While these problems have been conceptualized and written about at the architectural level, our specific goal is at the micro-level within a prexisting agent, LeafTutor. Thus, we rely on prompt-engineering literature to make our solutions to these established problems within prior research defensible. Reference Algorithm 1 for our pipeline pseudo-code. In it, let us consider P1 as an example of P within the final prompt to be inserted (line 21) within LeafTutor. For P1, we combine prompt theory with C (line 20). In this example, C is defined as encouragement and is the contextual solution to a problem L (line 19). L, in this instance (line 18), is the limitation of students rushing towards the answer and completing but not learning the assignment. P1 and other components of the final prompt are discussed in further detail later within this section. Furthermore, Figure 4, which identifies specific problems, their contextual solutions, prompt theory applied to those solutions, and final prompt creation, may be referenced throughout this section for specific prompt creation within the final prompt.

A. *Encouragement - P1*

One of the problems that can cause a student to rush towards an answer is students may experience anxiety and cognitive overload at the assignment. To solve this problem, we develop our AI persona to borrow from sports psychology used by elite athletes to do hard things [10]. With this approach we do three things.

One, we narrow the target, instructing the agent to break the task into one immediate objective. This sets an attainable goal and allows the student to focus on a small, completable step versus trying to mentally manage the entire assignment at once.

Two, we stabilize the student. We instruct the agent through our prompt that when frustration spikes to slow the pace, normalize difficulty and refocus attention.

Three, we reinforce the process cycle. We direct the agent that after each sub-goal to acknowledge completion and reset for the next manageable step.

Incorporating these three items within our prompt we aim to reduce anxiety and cognitive overload so that the student is comfortable to explore the topic and learn versus simply complete the assignment. As with any good tutor, learning is the goal.

Now that we have what we want to include in our prompt, we must lean upon prompt engineering theory to impose the above sentiment upon the agent. Our framework for the encouragement portion of the prompt has three particular items gleaned from prompt engineering theory [9]. Namely, we define the role of the agent (example seen in P1 Figure 4 as "You are...") define the task (seen as "break..." and "revisit..."), and define expected behavior and limits (seen as "monitor for..." and "without patronizing," for examples).

Thus, for the agent to behave as a tutor, focusing on learning versus assignment completion alone, we keep the student from rushing towards completion by borrowing from sports psychology and rooting it within a prompt engineering theory framework. This tailors the agent in a manner that direct student-LLM interaction would not achieve.

B. *Prerelease Prevention - P2*

With encouragement, we have addressed the student's rush towards the answer. Now, we will address the problem of the agent releasing the answer too soon. Thus, the student

Algorithm 1 Prompt-Engineering Pipeline for Enhancing Student Learning

function FINDLIMITATIONS(AITutorResearch)
    Identify core learning limitations addressable via prompting
    return L
end function
function DETERMINEPROMPTCONTEXT(L)
    Translate limitations into prompt-context requirements
    return C
end function
function DESIGNPROMPT(C, promptTheory)
    Construct internal prompt using promptTheory framework
    return P
end function
function INSERTFINALPROMPT(P, tutorAgent)
    Inject prompt into agent persona/guideline layer
end function

Main Pipeline
function ENHANCESTUDENTLEARNING(AITutorResearch, promptTheory, tutorAgent)
    $L \leftarrow$ FINDLIMITATIONS(AITutorResearch)
    $C \leftarrow$ DETERMINEPROMPTCONTEXT(L)
    $P \leftarrow$ DESIGNPROMPT(C, promptTheory)
    INSERTFINALPROMPT(P, tutorAgent)
end function

will remain in the learning phase depicted in Figure 4 for a longer period in order to understand the concepts better, further mimicking a good tutor.

The fundamental problem is without constraints the LLM may provide the final answer too early which will negatively affect the learning process [4].

As a solution, we provide explicit instructions to better guide the LLM to ensure learning is optimized for the student [16]. Specifically, we list things to avoid (e.g. solution), provide structure (e.g. prefer hints, prompt the student), and stay concise to keep on the topic. The framework for construction is provide a major constraint, suggested behavior, and guidance instructions.

Coupling the proposed solution with framework, we form P2 within Figure 4. Together with encouragement (P1 + P2), the student is well-prepared to focus on learning versus assignment completion, since the student is less likely to rush towards the answer, and the answer is constrained against prerelease. These solutions would be absent in raw student-LLM interaction.

C. Chat-free Directed Guidance - P3

With the student well-positioned for learning concepts, we focus on directed guidance and ease of interaction as the last component of prompt design (P3) in what we will refer to as a "chat-free" directed guidance option. With this chat-free feature, at the end of each response provided in the student-agent chat, three clickable options are listed to explore the topic more and further progress towards assignment completion.

The content of each of the three proposed responses are constructed with a "few-shot" framework within prompt engineering theory [16]. This approach argues not only giving instructions but explicitly demonstrating what one expects from the model can improve performance. Thus, within P3 (as seen in Figure 4), we direct the agent to provide three assistance options, yet also enumerate each one through three and provide an expectation of content for each.

Combining encouragement, prerelease constraints, and directed guidance in chat-free options (P1 + P2 + P3), this final prompt works as a constraint within the conceptual nervous system of our AI agent tutor to provide the student with an elevated experience not present in direct student-LLM interaction, complete with the ease of clickable "chat-free" options.

V. EXPERIMENT

For our experiment, we first define the metrics with which we will evaluate Leaf Tutor responses using the internal prompt constructed in the preceding section. Next, we implement a frame work to test the the agent using those metrics across multiple LLMs such as ChatGPT and Claude, obtain our results, and provide a discussion to interpret the findings.

A. Metrics

Given AI agent tutoring is an emerging field and standard metrics are still developing [7], [13], we design our own metrics based upon reasonable expectations of a human tutor in the approach detailed below.

The Leaf Tutor agent, by definition, seeks to mimic a good human tutor, assisting a student in a computer programming assignment. Imagining a human student-tutor interaction session,

one could monitor the session by "sitting in" and evaluating tutor responses and guidance based upon some fundamental expectations of tutor behavior. For example, a student visiting a good human tutor regarding an assignment would likely expect a discussion of concepts relevant to the assignment. Thus, it is reasonable to assume that a good human tutor could be evaluated during a tutor session on their ability to cover core concepts as well as their ability to provide optional concepts for depth. Since a student's time is valuable, one would also expect a human tutor to refrain from offering concepts unrelated to the task at hand and to provide efficient guidance in a brief manner. Thus, we may evaluate human tutor responses for irrelevance and efficiency. Furthermore, since computer science assignments can be mentally taxing on a student to begin with, it could reasonably be considered advantageous for a human tutor to deliver guidance in a naturally flowing manner, allowing the student to more easily grasp concepts with less distraction. Thus, a human tutor can be evaluated on the flow of their response delivery. Additionally, in a student-human tutor session, the student likely expects the tutor to cover a variety of topics, without needlessly repeating the same concepts over and over. Doing so would likely lose the student's attention, leading to a decreased desire to go see the tutor for help. Thus, an evaluator sitting in on a session may judge tutor responses on redundancy. Lastly, a student hopeful for assistance with an assignment certainly expects a human tutor to be able guide them through completion of the assignment within a reasonable amount of time. Thus, an evaluator may also wish to judge completion within the student-human tutor session.

Given these reasonable expectations of a human tutor, we evaluate our Leaf Tutor agent in the same manner with accuracy (core concept coverage), depth (optional concept coverage), noise (irrelevant topics introduced), efficiency, flow, redundancy, and completion, respectively. Instead of "sitting in" on a live student-human tutor session and marking human tutor responses, we will capture agent responses to standard questions on the student's behalf for evaluation.

The contribution of these metrics not only scores our ability to direct the Leaf Tutor through internal prompt design to ensure helpfulness akin to a human counterpart but also provides a pathway to evaluate the effectiveness of different LLMs such as ChatGPT and Claude within the agent.

1) Accuracy: We define accuracy to be the coverage of a predetermined set of core concepts related to a specific assignment. Core concepts can be defined as a set of concepts necessary for task completion. For example, let us consider an extremely simple assignment - write a program that takes two numbers and prints the sum. The set of core concepts, C, can be written to include: variable assignment, function definition, summation, return statement, and print statement. These concepts are considered necessary to complete the task. (A more in depth example of a predetermined set of core concepts relevant to a task can be seen in Figure 5.) If, over the course of the entire interaction with the agent regarding the assignment, the agent mentions only variable assignment and print statement, then only two of the five core concepts are covered. Thus, the agent's accuracy for the assignment in this example is 0.40. It is worth noting, this measurement is comprised of the sets of concepts. Repeats do not affect this metric. If the print statement concept was covered in multiple responses, for instance, the accuracy would remain 0.40. Figures closer to one indicate the agent covers more core concepts, while figures further from one indicate less core concept coverage. Formally, this metric can be defined as

$$\text{Accuracy} = \frac{|C_{\text{present}}|}{|C_{\text{total}}|}$$

where C_{present} represents the set of core concepts present during agent interaction related to an assignment, and C_{total} represents the full set of core concepts predetermined to be necessary for task completion.

2) Depth: We define depth to be a measure of how much further the agent goes above and beyond in explanation of the assigned task. To accomplish this, similar to how we defined core concepts above, we predetermine a list of optional concepts that the agent could cover that would be relevant to the task but not fundamentally necessary. Extending the very simple assignment example from above, a set of optional concepts can include: handling string inputs, returning a float variable, and rounding during the print statement. (A more in depth example of a predetermined set of optional concepts relevant to a task can be seen in Figure 5.) If during the course of interaction with the agent, only rounding was mentioned by the agent, then only one out of the three optional concepts are covered. Thus, the depth would be 0.33. As with accuracy, this metric is calculated using sets. Repeats, such as rounding appearing in multiple agent responses, do not affect the score. Likewise, as with accuracy, figures closer to one are more favorable because they indicate higher optional concept coverage. Depth can be formally defined as

$$\text{Depth} = \frac{|O_{\text{present}}|}{|O_{\text{total}}|}$$

where O_{present} represents the set of optional concepts identified in the responses, and O_{total} represents the full set of optional concepts.

3) Noise: Here we measure how much off-topic drift the responses provide. Specifically, we note the set of irrelevant topics covered during agent interaction while completing the assignment. In contrast with accuracy and depth, a set of irrelevant topics cannot be predetermined since it includes an infinite number of topics that may be deemed irrelevant to the task. Thus, the set of irrelevant concepts must be created and noted during agent interaction along the course of assignment completion. Continuing with our simple assignment, an example of an irrelevant topic would be connecting to a database. Clearly, this is not a necessary concept for task completion or an optional concept to provide further depth relative to the simple assignment. Thus, if this was the only irrelevant concept noted during responses for the entire assignment interaction with the agent, the set would include only one concept. For final computation, we would then divide

the number of irrelevant concepts, one in this case, by the union of the sets of predetermined core concepts, predetermined optional concepts, and noted irrelevant concepts. (A further example of noting irrelevant concepts pertaining to responses can be seen in Figure 5.) In short, it is the percentage coverage of irrelevant concepts relative to the total number of concepts covered during the course of the assignment. Thus, lower figures are preferred because they suggest the agent is not habitually mentioning the same concepts. Formally, we may write it as

$$\text{Noise} = \frac{|I_{\text{present}}|}{|C_{\text{present}} \cup O_{\text{present}} \cup I_{\text{present}}|}$$

where I represents the set of irrelevant (off-topic) concepts identified in the responses, C represents the set of core concepts present, and O represents the set of optional concepts present.

4) Efficiency: The goal of the efficiency metric is to determine how concise the responses are. Simply put, we wish to know how many relevant concepts are covered relative to the total word count. Recall our simple assignment example. The two core concepts of variable assignment and print statement are covered, and an optional concept of rounding is covered as well. Thus, there are three total relevant concepts present. If the total word count of all responses during agent interaction was thirty, then the efficiency score would be scored as three divided by 30, or 0.10. Larger figures are preferred, since that would mean more relevant concepts covered in fewer words. Given this is a ratio of concepts covered per word, we may obtain how many concepts are provided per ten words, as an example. In this instance, for concepts per ten words, we simply multiply the numerator and denominator by ten, resulting in one. Therefore, for this example, one concept is covered per every ten words, on average. We can further use this figure to compare efficiency between different LLMs used within the agent, such as ChatGPT and Claude. We can write the formula as such

$$\text{Efficiency} = \frac{|C_{\text{present}} \cup O_{\text{present}}|}{\text{WordCount}_{\text{all responses}}}$$

where C denotes the set of core concepts present in the response, O denotes the set of optional concepts present in the response, and Word Count represents the total number of words in all responses for that particular LLM.

5) Flow: This metric measures how well each response is internally put together and flows to the reader. Reviewing each response during agent interaction for assignment completion, if a response is manually judged as disjoint or confusing, it is scored as one. If a response is judged as somewhat clear, it is marked as two. If a response is judged to be smooth and logical, it is marked with a three. In the end, the measure of each response is averaged across all responses to give a flow score. Continuing with our above simple assignment example. If there are a total of twelve agent interactions (question-response pairs), four are scored as disjoint with a one, four are scored as somewhat clear with two, and four are scored as smooth and logical with three, then the flow score for the assignment would be 2.0 (four plus eight plus twelve, all divided by twelve total prompts). Higher figures for the score indicate greater flow for responses on average, while lower figures point towards less coherency. We may define this formula in the following manner

$$\text{Flow} = \frac{1}{k} \sum_{i=1}^{k} R_i$$

where R_i represents the flow score assigned to response i, and k represents the total number of responses evaluated.

6) Redundancy: For this metric, we are interested in how often concepts are repeated, with the assumption that similar responses habitually repeated by the tutor would be less helpful towards assignment completion. To accomplish this task, we take the total times each concept is mentioned across all responses (including duplicates) and subtract the number of unique concepts. This subtraction removes the first instance of the concept introduced, thus counting duplicates only. The difference is then divided by the total number of concepts mentioned. For instance, in our simple assignment example, let us consider the set of all concepts mentioned. This represents the total number of unique concepts and can be written as

$$N_{\text{unique}} = |C_{\text{present}} \cup O_{\text{present}} \cup I_{\text{present}}|$$

In our simple assignment example, we have a set of five core concepts, three optional concepts, and one irrelevant concept. The union of these sets gives us a value of nine. Thus, the number of unique concepts covered is nine. Let us say that across our total responses, the core concept of printing the statement was mentioned twice, and the optional concept of rousing was mentioned twice. All other concepts mentioned only occurred once. Thus, the total number of concept mentions would be eleven. Once we subtract nine from eleven, the difference is two. Thus, there were only two concepts repeated. We then divide this from the eleven concept mentions, giving us a ratio of repeats to total mentions of two to eleven or a redundancy score of 0.1818. The formula can be expressed as follows

$$\text{Redundancy} = \frac{N_{\text{total concept mentions}} - N_{\text{unique concepts}}}{N_{\text{total concept mentions}}}$$

where N_{total} represents the total number of concept mentions across all responses, and N_{unique} represents the number of unique concepts identified, defined as $|C_{\text{present}} \cup O_{\text{present}} \cup I_{\text{present}}|$.

7) Completion: Here we wish to measure if the agent was able to direct the user to completion, providing a fully working solution. To accomplish this we simply determine if the assignment was completed for each testing period in the following manner

$$\text{Completion} = \begin{cases} 1 & \text{if the assignment is successfully completed} \\ 0 & \text{otherwise} \end{cases}$$

where Completion indicates whether the agent successfully guided the user to a correct and complete solution to the assignment.

B. Implementation

To measure these features, we take the following steps. First, we select a simple coding assignment to upload to the AI tutor. Second, we evaluate the assignment and develop two lists, a core concept list which contains concepts we believe are fundamental towards completing the assignment and an optional concept list which contains concepts we believe could be helpful towards assignment completion but are not fundamentally necessary. These lists will be necessary for calculations shown later within this section.

Next, we decide a number of prompts (k) that denotes a reasonable ceiling in which the agent should have helped the user complete the assignment. For each step toward solving the assignment (1-k), we then develop a fixed sequence of questions (Q1–Qk) that will be used identically across both LLMs we are testing.

For each LLM to be tested for each question (Q1-Qk) to be asked, we develop a score sheet. (See Figure 5.) Each score sheet contains the following: LLM-type (OpenAI or Anthropic), question (Qi), response (Ri), the list of predetermined core concepts in checkable boxes, the list of predetermined optional concepts in checkable boxes, a location for write-in irrelevant concepts mentioned, a flow score (1-3) for the response (Ri) (1 = disjoint/confusing; 2 = somewhat clear; 3 = smooth logical), a concise score (1-3) for the response (Ri) (1 = overly verbose, 2 = mostly concise; 3 = efficient and tight), finally, a word count for each response (Ri).

The appropriate LLM is selected (OpenAI or Anthropic) and noted, and the assignment is then uploaded into the AI tutor following the process depicted in Figure 1. The fixed questions (Q1-Qk) are asked in order without deviation and the corresponding responses (R1-Rk) are then recorded verbatim on each score sheet.

Finally, it is noted whether the agent was able to complete the assignment, given the questions (Q1-Qk).

Then, the process is repeated for the remaining LLM from the OpenAI, Anthropic choice set. Once we have our completed score sheets for each LLM and each fixed question (Q1-Qk), we are ready to compute our desired metrics.

C. Results

This section evaluates the performance of two large language models, Anthropic (Claude) and OpenAI, within the LeafTutor framework using the evaluation metrics defined in Section VII.

1) Experimental Setup: Evaluation was conducted on three Python programming assignments of increasing complexity: A1 (*Distance Between Points*), requiring Euclidean distance computation and list iteration; A2 (*Word Frequency Counter*), involving string processing, punctuation removal, and dictionary construction; and A3 (*Grade Summarizer*), requiring dictionary filtering, averaging, and decimal rounding. Seven core concepts and six optional concepts were identified for each assignment. Responses from both AI services were given to questions posed in sequence followed by requests to modify codes.

Table I summarizes the seven evaluation metrics, their formulas, and interpretation directions. These metrics collectively capture concept coverage (Accuracy, Depth), content relevance (Noise), informativeness per word (Efficiency), pedagogical sequencing (Flow), concept reuse (Redundancy), and task outcome (Completion).

Accuracy and Depth determine accuracy of coverage. Conceptual noise is measured by pollution level. Efficiency gives credit to agents using less text to communicate a higher number of concepts. Flow determines the efficiency of teaching sequence per response. Redundancy determines the frequency with which an agent revisits previously discussed concepts; excessive redundancy can be attributed to either repetitive explanation method or strategic repetition using full code samples.

2) Per-Assignment Results: Table II reports all seven metrics for both models across the three assignments, together with cross-assignment averages. Bold entries indicate the per-cell winner. Depth is included for completeness; both models tied at 66% in every assignment, confirming that optional concept coverage was driven by the fixed question sequence rather than model capability.

Table III consolidates the per-assignment winners for each metric and identifies the overall winner. Claude wins four of seven metrics; OpenAI wins two (Efficiency and Redundancy). One was tie (Depth).

Table II summarizes the performance of both models across all evaluation metrics. Claude was more reliable pedagogically than OpenAI in the three assignments; it never failed to hit any of the necessary concepts, brought the smallest amount of irrelevant information, and generated the best quality of response flow, particularly in code revision assignments where diagnosing the issue and correcting the code meant the difference between success and failure for the students. However, the strengths OpenAI showed in Efficiency and Redundancy are linked to its ability to generate responses that were more concise, although at a cost of increased noise and less accurate corrections in code assignments.

Accuracy: Claude was able to cover all core concepts 100% in all three tasks. OpenAI scored equally well in terms of accuracy in A1 and A3, but failed to introduce a return statement in A2, marking the only such deviation across the entire study. Both Claude and OpenAI managed to cover 4 out of 6 optional concepts in each assignment, scoring equally well with 67% in Depth.

Depth: Each of the providers was able to cover exactly 4 out of 6 optional enrichment concepts in each assignment, thus having an equal score of 67% in the Depth dimension for all three assignments, which is the only dimension in which there is no advantage to either LLM. Claude and OpenAI did not address issues such as dealing with an empty list as input, handling edge cases, and sorting keys in the output at any point in their response chains. The fact that the two providers were tied on the issue indicates that the issue of covering optional

TABLE I
EVALUATION METRIC DEFINITIONS, FORMULAS, AND INTERPRETATION DIRECTIONS.

Metric	Definition	Formula	Direction
Accuracy	Core concept coverage	$\lvert C^*_{\text{present}}\rvert \;/\; \lvert C_{\text{required}}\rvert$	↑
Depth	Optional concept coverage	$\lvert O^*_{\text{present}}\rvert \;/\; \lvert O_{\text{possible}}\rvert$	↑
Noise	Irrelevant concept ratio	$\lvert I^*\rvert \;/\; (\lvert I^*\rvert + \lvert C^*\rvert + \lvert O^*\rvert)$	↓
Efficiency	Useful concepts per word	$(\lvert C^*\rvert + \lvert O^*\rvert) \;/\; W_{\text{total}}$	↑
Flow	Avg. response quality (1–3)	$\sum_{i=1}^{k} f_i \;/\; k, \quad f_i \in \{1,2,3\}$	↑
Redundancy	Repeated-concept mention rate	$(N_{\text{total}} - \lvert U\rvert) \;/\; N_{\text{total}}$	↓
Completion	Task completion (binary)	1 if fully solved, 0 otherwise	↑

*Unique. C = core, O = optional, I = irrelevant; W_{total} = total words; k = responses.

TABLE II
PER-ASSIGNMENT METRIC RESULTS FOR CLAUDE (C) AND OPENAI (O). **BOLD** = WINNER PER CELL. ↑ HIGHER IS BETTER; ↓ LOWER IS BETTER.

Assign.	Accuracy ↑		Depth ↑		Noise ↓		Flow ↑		Efficiency ↑		Redundancy ↓		Completion ↑	
	C	O	C	O	C	O	C	O	C	O	C	O	C	O
A1	**100%**	100%	66%	66%	**8%**	15%	**2.67**	1.71	0.016	**0.017**	56%	**46%**	**Yes**	No
A2	**100%**	86%	66%	66%	**0%**	9%	**2.80**	2.38	0.011	**0.015**	73%	**54%**	**Yes**	No
A3	**100%**	100%	66%	66%	**8%**	21%	**3.00**	2.50	0.012	**0.012**	66%	**58%**	**Yes**	No
Avg.	**100%**	95%	66%	66%	**6%**	15%	**2.82**	2.20	0.013	**0.015**	65%	**53%**	**3/3**	0/3

TABLE III
PER-METRIC WINNERS BY ASSIGNMENT AND OVERALL. **C** = CLAUDE, **O** = OPENAI.

Metric	A1	A2	A3	Overall	Dir.
Accuracy	Tie	**C**	Tie	**C**	↑
Depth	Tie	Tie	Tie	Tie	↑
Noise	**C**	**C**	**C**	**C**	↓
Efficiency	**O**	**O**	**O**	**O**	↑
Flow	**C**	**C**	**C**	**C**	↑
Redundancy	**O**	**O**	**O**	**O**	↓
Completion	**C**	**C**	**C**	**C**	↑

concepts depends mainly on the question set and not on the provider's competence.

Noise: Claude delivered a noticeably lower noise level than OpenAI in all three tasks (average: 5.6% vs 15.6%). The biggest difference was observed in Task 3, where OpenAI showed noise of up to 21%. This was due to an incorrectly written variable name in OpenAI's sample (`result[student_name]`), which was directly copied by the student and resulted in a bug. In Task 2, OpenAI introduced the concept of list comprehension without request. Claude provided some irrelevant information in A1 (list slicing) and A2 (mentioning test framework).

Flow: The average flow scores for Claude in both A1 and A2 were higher than those of OpenAI. In addition, while Claude's answers were slightly higher in terms of flow in A3 (3.00 vs. 2.50), OpenAI's answers had repetitive patterns in all three assignments. In A1, there was a repetition of a concept already discussed by the teacher (A1-R3, A1-R5), and OpenAI was unable to provide any code revision answers. On the contrary, Claude succeeded in revising the code in all three assignments with detailed answers.

Efficiency: OpenAI produced more concepts per word in all three assignments (means: 0.0147 vs. 0.0128). This benefit results from OpenAI producing shorter overall answer length, 675 words in A2 vs. 1,030 by Claude, and not repeating whole code blocks. Despite this increased efficiency, it needs to be taken together with the Noise parameter, since some of OpenAI's reduced length was achieved at the expense of missing important information and errors in critical answers.

Redundancy: Claude demonstrated higher redundancy than OpenAI in all assignments (means: 65.1% vs. 52.6%). The higher level of redundancy of Claude refers to the fact that the percentage of the concepts mentioned was higher because the concepts had been repeated in many answers. This fact may be explained by the fact that Claude tends to insert whole code blocks into further answers, thus mentioning the same set of concepts again.

Completion: It is notable that in all three assignments, OpenAI was unable to lead students to a successful solution, although Claude succeeded fully in all three assignments. The disparity between the two AI assistants in terms of their contribution to student progress at 100% vs. 0% is by far the most important point in the research, as it confirms that the Flow and Noise metrics for Claude positively contributed to the results obtained by students, unlike OpenAI, which did not provide correct solutions to programming problems.

D. Discussion

The results indicate that Claude provides more effective tutoring support within the LeafTutor framework. Its strengths include higher conceptual coverage, stronger alignment with instructional targets, and more consistent interaction flow. Additionally, Claude completed all stages of the assignment, while OpenAI failed to address critical intermediate steps in the problem-solving process.

Although OpenAI demonstrated reasonable performance on simpler conceptual tasks, it struggled with maintaining continuity and providing complete guidance in multi-step scenarios. These findings highlight the importance of both model capability and system-level constraints in determining the effectiveness of AI-driven tutoring systems.

Overall, the results support the conclusion that structured prompt engineering, when combined with a capable underlying model, can significantly improve both the quality and educational value of AI-assisted tutoring systems.

VI. Future Directions

A promising direction for future work is the integration of adversarial and multi-agent LLM frameworks to enhance the quality, reliability, and pedagogical effectiveness of the tutoring system. Current implementations typically rely on a single model to generate instructional responses, which can result in issues such as hallucinated explanations, incomplete reasoning, or pedagogically suboptimal guidance [3].

One approach to addressing these limitations is the use of a *generator-critic* architecture, in which one model produces a tutoring response and a second model evaluates its correctness, clarity, and instructional value. This paradigm is closely related to recent work on multi-agent debate systems, where multiple language models iteratively critique and refine outputs to improve factual accuracy and reasoning quality [17]. In the context of programming education, such a system could ensure that generated explanations are not only syntactically correct but also conceptually sound and aligned with best practices.

Additionally, adversarial prompting techniques can be used to systematically identify weaknesses in the tutoring agent by generating challenging or misleading student inputs [18]. These inputs may include ambiguous problem statements, incorrect assumptions, or edge-case code scenarios. Incorporating such adversarial examples into evaluation pipelines would enhance the system's robustness and reduce the likelihood of propagating incorrect or misleading guidance.

Another important extension involves the incorporation of automated feedback loops inspired by reinforcement learning from human feedback (RLHF), where model outputs are iteratively refined based on evaluative signals [19]. In a tutoring setting, this could be adapted to prioritize responses that encourage critical thinking, provide hints rather than direct answers, and scaffold learning in a manner consistent with established educational practices [20].

Finally, future systems may benefit from integrating persistent memory and student modeling, allowing the agent to adapt explanations based on a learner's prior interactions and demonstrated skill level. Such personalization is a core principle of intelligent tutoring systems and has been shown to significantly improve learning outcomes [20]. Combined with adversarial evaluation and multi-agent refinement, this would enable the development of more adaptive, reliable, and pedagogically effective AI tutoring systems.

Overall, the incorporation of adversarial and multi-agent LLM architectures, alongside principles from intelligent tutoring systems, represents a promising pathway toward more reliable, interpretable, and educationally effective AI-driven programming tutors.

VII. Conclusion

This paper introduced LeafTutor, an AI agent designed to support programming education by integrating large language models with structured prompt engineering and agent-based control mechanisms. The primary goal was to shift AI-assisted learning from solution generation toward guided conceptual understanding.

The results demonstrate that model behavior varies significantly within a constrained tutoring framework. Anthropic's Claude consistently outperformed OpenAI across key metrics, including Accuracy, Depth, Flow, and Completion. These differences were most pronounced in multi-step problem-solving scenarios, where Claude maintained stronger continuity and introduced more comprehensive conceptual coverage.

The findings reinforce two central conclusions. First, the effectiveness of AI tutoring systems depends not only on the underlying language model but also on how that model is structured and constrained within an agent framework. Second, prompt engineering plays a critical role in shaping model behavior, particularly in educational settings where learning progression and conceptual clarity are prioritized.

While the proposed evaluation framework provides meaningful insight into model performance, it is limited by its reliance on keyword-based concept extraction and aggregated scoring methods. Future work may explore more advanced semantic evaluation techniques, as well as the integration of multi-agent and adversarial LLM architectures to further improve response quality and reliability.

In conclusion, this work demonstrates that combining LLM capabilities with structured agent design and pedagogically informed interaction strategies can significantly improve the effectiveness of AI-driven programming tutors. As AI continues to evolve, such approaches will be essential in ensuring that these systems support meaningful learning outcomes rather than simple task completion.

References

[1] A. Vaswani *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[2] T. B. Brown *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, 2020.

[3] R. Bommasani *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[4] M. B. Mutanga, "Exploring the impact of llm prompting on students' learning," *Trends in Higher Education*, vol. 4, no. 3, 2025.

[5] A. Alkhatlan and J. K. Kalita, "Intelligent tutoring systems: A comprehensive historical survey with recent developments," *International Journal of Artificial Intelligence in Education*, vol. 29, pp. 1–29, 2018.

[6] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[7] D. Weitekamp, M. N. Siddiqui, and C. J. MacLellan, "Tutorgym: A testbed for evaluating ai agents as tutors and students," in *Lecture Notes in Computer Science*. Springer, 2025.

[8] J. D. Karpicke and J. R. Blunt, "Retrieval practice produces more learning than elaborative studying with concept mapping," *Science*, vol. 331, no. 6018, pp. 772–775, 2011.

[9] J. D. Velásquez-Henao, C. J. Franco-Cardona, and L. Cadavid-Higuita, "Prompt engineering: A methodology for optimizing interactions with ai-language models in the field of engineering," *DYNA*, vol. 90, no. 230, pp. 9–17, 2023.
[10] D. Gould and I. Maynard, "Psychological preparation for the olympic games," *Journal of Sports Sciences*, vol. 27, no. 13, pp. 1393–1408, 2009.
[11] G. Jošt, V. Taneski, and S. Karakašić, "The impact of large language models on programming education and student learning outcomes," *Applied Sciences*, vol. 14, no. 11, p. 4115, 2024.
[12] C.-H. Lai and C.-Y. Lin, "Its-cal: Intelligent tutoring system for coding and learning," *Applied Sciences*, 2025.
[13] W. Lyu *et al.*, "Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study," 2024.
[14] M. Müller, C. List, and M. Kipp, "The power of context: An llm-based programming tutor for beginning programming students," in *Proceedings of the 6th European Conference on Software Engineering Education*, 2025.
[15] A. W. Kruglanski and D. M. Webster, "Motivated closing of the mind: "seizing" and "freezing"," *Psychological Review*, vol. 103, no. 2, pp. 263–283, 1996.
[16] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.
[17] Y. Du *et al.*, "Improving factuality and reasoning in language models through multi-agent debate," *arXiv preprint arXiv:2305.14325*, 2023.
[18] E. Perez *et al.*, "Red teaming language models with language models," *arXiv preprint arXiv:2202.03286*, 2022.
[19] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, 2022.
[20] J. R. Anderson *et al.*, "Intelligent tutoring systems," *Science*, vol. 268, no. 5210, pp. 895–902, 1995.

Fig. 4. Step-by-step prompt construction

LLM Type: OpenAI

Question (Q1): [Insert question 1 from the pre-determined list of questions Qi-Qk here]

Response (R1): (Paste full response from the AI tutor below)

Concept Identification: Circle each that applies. Concepts must be selected from the predefined lists. Similar phrasing should be mapped to the same concept for consistency across responses. If you note an irrelevant concept, write it in (2-3 max).

Core Concepts	Optional Concepts	Irrelevant Concepts (max 3)
Function definition (word frequency): defines function taking text and returning result	Handling empty input: describes behavior for empty/null input	1. _____
Case normalization: converts text to lowercase	Handling extra whitespace: addresses irregular spacing	2. _____
Punctuation removal: removes or ignores punctuation	Iteration over words: loops through tokens	3. _____
Word tokenization: splits text into words	Code readability/structure: clear variables and logic	
Dictionary usage: stores word-frequency pairs	Edge case discussion: explains handling of non-standard inputs	
Frequency counting: increments counts for repeated words	Test cases/examples: provides sample inputs/outputs	
Return statement: returns final dictionary		

Response Flow Score (1–3): ____

(1 = disjoint/confusing, 2 = somewhat clear, 3 = smooth/logical)

Response Conciseness Score (1–3): ____

(1 = overly verbose, 2 = mostly concise, 3 = efficient/tight)

Response Word Count: ____

Fig. 5. Example score sheet for each LLM used and each Qi-Ri question-response.