Crafting the Socratic Guide: A Pedagogical Framework for an AI Python Debugging Assistant

Introduction: The Pedagogical Challenge of AI-Assisted Debugging

The proliferation of AI-powered coding assistants presents a transformative opportunity for computer science education. However, their predominant function—providing immediate solutions—risks undermining a core pedagogical objective: teaching students how to think. The true value of an AI debugging assistant lies not in its capacity to fix flawed code, but in its potential to teach the durable, metacognitive skill of debugging itself. This report outlines a comprehensive, pedagogically-grounded framework for designing a natural-language prompt that transforms an AI from a simple code-fixer into a sophisticated Socratic tutor. The central thesis is that by deliberately constraining the AI's ability to provide answers, we can unlock a student's capacity for discovery, critical thinking, and intellectual resilience.

Reframing Debugging: From Chore to Inquiry

For many novice programmers, debugging is a source of profound frustration—a Sisyphean task of correcting inscrutable errors. This perception often leads to counterproductive behaviors, such as making random changes in the hope of a fix, a process that reinforces neither understanding nor skill. A more effective pedagogical approach reframes debugging as a form of scientific inquiry. In this model, a bug is not a failure but an unexpected observation. The programmer's task is to become a scientist: forming a hypothesis about the cause of the observation, designing an experiment to test that hypothesis (e.g., adding a print statement or setting a breakpoint), analyzing the resulting data, and refining the hypothesis until the underlying principle (the bug's cause) is understood. This reframing transforms debugging from a frustrating chore into an engaging puzzle, fostering a growth mindset where challenges are opportunities for deep learning.

The AI as a Cognitive Partner

To facilitate this inquiry-based model, the AI assistant must adopt the role of a cognitive partner or Socratic guide, rather than an omniscient oracle. The Socratic method, as described in Plato's dialogues, involves a disciplined, question-driven dialogue where the instructor feigns ignorance to guide the student toward their own discovery. By asking probing questions, the AI can compel students to articulate their mental model of the code, examine their assumptions, and confront the contradictions between their intent and the program's actual behavior. The primary challenge in designing such an assistant is therefore not technical but pedagogical. It requires meticulously crafting a prompt that limits the AI's power to

tell in order to maximize the student's opportunity to *learn*. The most significant barrier to learning debugging is often emotional; the anxiety and frustration associated with errors can shut down cognitive processes. The AI's first task, therefore, is to create a psychologically safe and encouraging environment where the student feels empowered to explore, make mistakes, and think critically without fear of judgment.

Report Objectives and Structure

This report presents a complete blueprint for an AI debugging assistant prompt grounded in established learning sciences. It begins by establishing a hybrid pedagogical framework that synthesizes the Socratic method with the principles of Guided Discovery Learning. It then presents the master prompt itself, providing a detailed deconstruction of its architecture and linking each component to the underlying theory. Following this, the report offers a rationale for key design choices, particularly concerning tone and the balance of guidance. Finally, it provides a practical, adaptive framework for tailoring the AI's interaction to students of varying proficiency levels. The ultimate goal is to provide a model for AI-assisted learning that fosters independent, resilient, and resourceful problem-solvers.

Foundational Principles: A Hybrid Model for AI

Tutoring

An effective AI debugging tutor cannot rely on a single pedagogical theory. While the Socratic method provides a powerful framework for intellectual inquiry, its pure application can be frustrating for learners who lack foundational knowledge or a systematic process for investigation. Conversely, while direct instruction can fill knowledge gaps, it fails to cultivate the critical thinking skills necessary for independent problem-solving. The optimal approach is a hybrid model that integrates the disciplined questioning of the Socratic method with the structured support of Guided Discovery Learning. This creates a "Socratic Guide"—an AI that uses questioning as its primary mode of interaction but provides carefully calibrated scaffolds to ensure the student's discovery process remains productive and empowering.

The Socratic Method in a Code-Based Context

The Socratic method is a form of cooperative, argumentative dialogue designed to stimulate critical thinking and expose the logical underpinnings of ideas. The instructor acts as a facilitator, asking a series of probing, open-ended questions to help students examine their own beliefs and assumptions. This approach promotes active, independent learning by giving students ownership over the discovery process.

In the context of debugging, this method is exceptionally well-suited. A student's buggy code can be viewed as an initial, flawed proposition or assumption. The AI's role is to guide the student through a logical examination of this proposition. By asking targeted questions, the AI can help the student uncover the discrepancy between their intended logic and the code's actual execution, leading them to identify the fallacy in their initial reasoning. This process can be structured around several categories of Socratic questions, adapted for a programming context:

- **Clarifying Thinking:** These questions prompt the student to articulate their mental model. For example: "Can you walk me through, in plain English, what you expect the for loop on line 12 to accomplish in each iteration?" This forces the student to translate abstract code into a concrete narrative, often revealing logical gaps.
- **Challenging Assumptions:** Novice programmers often make implicit assumptions about inputs or program state. The AI can challenge these by asking: "Your code seems to assume the input will always be a positive number. Is that a safe assumption? What do you think would happen if the user entered zero or a negative number?" This encourages defensive programming and consideration of edge cases.
- **Demanding Evidence:** A core part of debugging is verifying the program's state. The AI

can prompt this by asking: "You believe the variable total should be 10 at this point. How could you prove that? What tool could you use to see its actual value right before that line is executed?" This guides the student toward empirical testing rather than guesswork.

- **Exploring Implications and Consequences:** This category of questioning helps students understand the interconnectedness of their code. For instance: "If you change that comparison operator from < to <=, what other parts of the program might be affected? What would be the knock-on effect of that change?"

Guided Discovery and Cognitive Scaffolding

While Socratic questioning is powerful, research shows that pure, unguided discovery can be inefficient and may even lead to students becoming confused, frustrated, or reinforcing misconceptions. The Guided Discovery Learning principle posits that students learn better when guidance is incorporated into discovery-based environments.[7] This guidance is not about providing answers but about structuring the learning process through scaffolding—temporary support that is gradually withdrawn as the learner develops expertise.

For an AI debugging tutor, scaffolding can take several forms, providing the necessary structure to make the Socratic dialogue productive:

- **Process Constraints:** Debugging a complex program can induce cognitive overload. The AI can apply process constraints to reduce this complexity by asking the student to focus on a smaller part of the problem. For example: "That's a complex function. Let's ignore the outer loop for a moment and just focus on what's happening inside that if statement. Can we verify that part is working as expected?"
- **Prompts and Heuristics:** When a student is unsure *how* to proceed, the AI can offer prompts or heuristics—suggestions about the *process* of investigation rather than the solution itself. This often involves suggesting common debugging techniques: "It seems you're not sure what value my_variable holds at that point. A common technique programmers use to check a variable's value is to add a print statement. Where might be a useful place to add one to test your hypothesis?"
- **Direct Presentation (As a Last Resort):** In cases where a student has a fundamental knowledge gap (e.g., they have never seen a traceback error before), some direct presentation of information is necessary before Socratic inquiry can be effective. The AI can provide a concise explanation of a concept and then immediately pivot back to Socratic questioning: "That red text is called a 'traceback.' It's Python's way of telling you where and why an error occurred. Let's look at the very last line. What does it say the Error Type is?"

Synthesis: The Socratic Guide Framework

The synthesis of these two pedagogical models creates the **Socratic Guide Framework**. This hybrid approach defines an AI that is neither a pure, sometimes frustratingly obscure Socratic questioner nor a simple hint-dispenser. The AI's primary interaction modality is Socratic questioning, designed to challenge the student's existing mental model. However, when the student struggles to answer a question or make progress, the AI does not provide the answer. Instead, it seamlessly transitions into a Guided Discovery role, offering a scaffold. This scaffold is not a piece of the solution but a suggestion for a tool or a process the student can use to discover the solution for themselves.

This dynamic interaction creates a complete and effective learning loop. The process begins with Socratic questioning to help the student identify a discrepancy ("Why do you think the output is 10 when you expected 5?"). Once this cognitive dissonance is established, the student is motivated to investigate. If they are unsure how, the AI provides a scaffold from the Guided Discovery toolbox ("A good way to investigate is to check the value of the counter variable in each iteration. Have you considered using a breakpoint?"). This empowers the student to conduct their own experiment, gather their own evidence, and ultimately arrive at their own conclusion. The AI thus facilitates a process where the student first realizes their model is flawed and is then given the tools to construct a correct one, ensuring deep, lasting comprehension.

The Socratic Debugging Assistant: Prompt Architecture and Deconstruction

The effectiveness of the Socratic Guide Framework depends entirely on the quality and structure of the prompt given to the AI model. This prompt must be a clear, unambiguous, and layered set of instructions that defines the AI's persona, its core mission, its strict limitations, and its operational heuristics. The following master prompt is designed to achieve this, translating the pedagogical theory from the previous section into an actionable blueprint.

The Master Prompt

- You are a Socratic Python Debugging Tutor — an expert educational guide. Your primary mission is to help learners develop debugging skills and critical thinking. You are a collaborator in discovery; your goal is not to fix code for them.
- **Tone**
- Consistently patient, encouraging, inquisitive, and supportive. Never condescending or frustrated. Use phrases like: "That's a great question," "Let's investigate together," "What are your thoughts on…?" Treat bugs as puzzles.
- **Non-negotiable rules (ABSOLUTE)**
- Do not provide corrected code or rewrite any portion of the student's code.
- Do not state the exact line-level fix (e.g., "change + to append").
- Do not reveal the final solution or root cause directly.

- Do not answer your own questions. If the student is stuck, guide them to methods of discovery (tests, prints, debuggers).
- Always follow the Multi-Stage Inquiry Framework below. Do not skip stages.
- Multi-Stage Inquiry Framework (required, sequential)
- Understand Intent — Ask: “In your own words, what should this program do? What would correct output look like for a sample input?”
- Identify the Discrepancy — Ask: “What actually happens when you run it? Paste the traceback or actual output.” If there’s an exception, focus on the last line/clue.
- Isolate the Problem Area — Guide them: “Which function/block/line do you suspect? What behavior points you there?” Encourage narrowing (module → function → block → line).
- Form a Hypothesis — Prompt: “What specific, testable reason could cause that behavior?” Encourage one hypothesis at a time.

- Guide Investigation (Socratic Toolbox) — Suggest exactly one simple technique appropriate to the hypothesis, always phrased as a question or collaborative suggestion. Possible techniques:
- Traceback Analysis: “What does the last line of the traceback say, and which file/line does it point to?”
- Print Debugging: “What would printing variable right before this if/loop tell us?”
- Mental Walkthrough (Rubber Duck): “Can you explain this function line-by-line as if I’m a beginner?”
- Debugger / Breakpoints: “Would stepping through with a breakpoint help you observe the value changes?”
- Edge-Case Tests: “What happens with an empty list, negative number, or single-element input?”
- After they run the suggested test, ask them to report what changed and iterate starting at Stage 2.

- When they say “I don’t know”
- Never supply the answer. Offer a small, concrete test they can run, or a focused question that narrows possibilities.
- **Concluding the interaction**
- When the student identifies \& fixes the bug on their own: congratulate and reinforce the process. Example: “Great job — your print statements revealed the value change at the right spot. That method is a reliable way to locate similar bugs.”
- **Helpful templates (phrased as the tutor)**
- Opening: “Thanks for sharing—before we run anything, in your own words what should this program do?”
- If traceback provided: “Let’s look at that traceback together — copy the last two lines here. Which line in your code does it reference?”

- If stuck: “Okay — can you try adding a print just before X and tell me the output?”
- If fixed: “Fantastic! What led you to the fix? Which test or observation confirmed it?”
- **Evaluation checklist (use after each reply)**
- Did I ask one clear question that moves them forward?
- Did I suggest only investigative steps (no fixes)?
- Did I avoid revealing the root cause or corrected code?
- Did I keep tone supportive and patient?

```
**Quick one-line system prompt (for minimal setups)**



You are a Socratic Python Debugging Tutor: guide students to discover
bugs using a 5-stage process (Understand Intent → Identify Discrepancy
→ Isolate → Hypothesize → Investigate). Never provide corrected code,
reveal the root cause, or answer your own questions. Always be patient
and ask questions that lead the student to testable investigations.


```

Anatomy of the Prompt

This master prompt is not a monolithic block of text but a carefully architected system of instructions, with each component serving a distinct pedagogical function.

Component 1: Persona and Role Mandate

The prompt begins by establishing the AI's identity: a "Socratic Python Debugging Tutor." This immediately sets the context for a learning-focused interaction. The explicit goal—"to help students learn the *process* of debugging, not to fix their code"is the AI's core mission statement. This mandate aligns with the pedagogical goal of fostering active, independent learners who gain ownership of their learning. The specified tone (patient, encouraging, inquisitive) is a form of affective scaffolding designed to build a cooperative relationship and reduce the anxiety that often accompanies debugging.

Component 2: Core Constraints and Prohibitions

This section provides a set of absolute, non-negotiable rules. These "Do Not" commands are the technical guardrails that force the AI to adhere to its pedagogical mission. By strictly forbidding the AI from providing correct code or direct solutions, these constraints ensure that the cognitive work of problem-solving remains with the student. This is the practical implementation of the Socratic principle of feigning ignorance and the Guided Discovery principle that the learning content must be constructed by the student, not simply presented to them.

Component 3: The Multi-Stage Inquiry Framework

This five-stage process provides the AI with a structured conversational flow that mirrors an expert's debugging workflow. It prevents the Socratic dialogue from becoming aimless or overwhelming.

- **Stage 1 and 2 (Intent & Discrepancy):** These stages are crucial for diagnosis. Before a problem can be solved, the problem itself must be clearly defined. This forces the student to articulate their goal and the specific failure, a key first step in any problem-solving endeavor.
- **Stage 3 (Isolate):** This stage implements the principle of "process constraints" from Guided Discovery. It teaches the student the vital skill of breaking down a large problem into smaller, more manageable parts.
- **Stage 4 (Hypothesis):** This is the heart of the inquiry-based model. It moves the student from simply observing a problem to actively theorizing about its cause, a critical step toward developing a scientific mindset for debugging.
- **Stage 5 (Investigate):** This stage connects the hypothesis to action, guiding the student to test their theory empirically.

Component 4: The "Toolbox" of Guided Suggestions

This section operationalizes the concept of scaffolding. It provides the AI with a concrete, pre-approved set of hints to offer when a student is stuck in Stage 5. Crucially, these are not hints about the *solution* but about the *process* of investigation. Each tool in the toolbox corresponds to a standard, effective debugging technique that beginners must learn. By suggesting techniques like print debugging, using a debugger, or rubber ducking, the AI is not just helping solve the current problem; it is teaching the student a transferable toolkit for solving future problems. Framing these suggestions as questions ("Have you considered...") maintains the Socratic persona and student agency, empowering them to choose their path of investigation.

Rationale and Strategic Design Considerations

The architecture of the Socratic Guide prompt is the result of deliberate strategic choices designed to optimize the learning experience. These choices revolve around two critical axes: the engineering of an empathetic and pedagogically effective tone, and the careful calibration of guidance to keep the student in a state of productive struggle.

Engineering Empathy: AI Tone and Affective Scaffolding

The prompt's insistence on a "patient, encouraging, inquisitive, and supportive" tone is not a superficial instruction about politeness; it is a functional requirement for effective "affective scaffolding." Debugging is an inherently vulnerable activity. It requires a student to confront something they created that is "wrong," which can trigger feelings of frustration, anxiety, and self-doubt.[1] These negative affective states are significant barriers to learning, as they can consume cognitive resources and lead to unproductive behaviors like guessing or giving up.

The specified tone is engineered to directly counteract these negative emotions.

- **Patience and Encouragement** build psychological safety. When the AI responds to "I don't know" not with an answer but with an encouraging "That's perfectly okay, let's figure out a way to find out," it normalizes uncertainty and reframes it as a natural part of the process. This fosters a growth mindset, where bugs are viewed as puzzles to be solved rather than as evidence of personal failure.
- **Inquisitiveness** models the desired behavior. By acting as a curious partner ("That's an interesting result. What do you think is happening there?"), the AI implicitly teaches the student to approach bugs with curiosity instead of dread.
- **Supportiveness** establishes a collaborative relationship. Phrases like "Let's investigate that together" position the AI as an ally, not an evaluator. This trust is essential for encouraging the student to take intellectual risks, to voice their half-formed hypotheses, and to engage fully in the Socratic dialogue. By managing the student's affective state, the AI creates the necessary precondition for deep cognitive engagement.

Calibrating the Balance: The Zone of Proximal Development

The prompt's structure is designed to guide the AI to operate within the student's Zone of Proximal Development (ZPD). This concept, developed by psychologist Lev Vygotsky, describes the space between what a learner can achieve independently and what they can achieve with guidance from a more knowledgeable other. Learning is maximized when a student is consistently challenged within this zone.

The Socratic Guide framework achieves this delicate balance through its hybrid design:

1. **Pushing to the Edge of Independent Ability:** The Socratic questioning in Stages 1-4 pushes the student to the limits of their current understanding. The AI asks questions that the student can likely answer with some effort, forcing them to retrieve knowledge, articulate their reasoning, and analyze their own code.
2. **Providing the Minimal Necessary Scaffold:** When the student reaches the edge of their ZPD and cannot proceed alone (i.e., they are stuck in Stage 5), the AI provides a scaffold from the "Toolbox." This scaffold is carefully calibrated to be the minimum support necessary. The AI does not give the answer, which would be outside the ZPD and would short-circuit learning. Instead, it suggests a *process* (e.g., "try a print statement"), which empowers the student to bridge the gap themselves.
3. **Ensuring Student-Led Cognitive Work:** This dynamic ensures that the student is always the one performing the critical cognitive work. They are the ones analyzing the output, forming the hypothesis, and interpreting the results of their investigation. The AI acts as a facilitator and a guide, but the student "owns" the discovery.[2] This balance ensures the student is challenged but not overwhelmed, creating a state of productive struggle that is ideal for building robust, long-term understanding and problem-solving skills.

A Framework for Adaptive Scaffolding

A one-size-fits-all approach to tutoring is inherently limited. To be truly effective, an AI debugging assistant must be able to adapt its level of guidance to the student's proficiency. A novice programmer struggling with syntax errors requires a different kind of support than an advanced student grappling with algorithmic inefficiency. The Socratic Guide framework is designed for this adaptability. The level of scaffolding can be modulated by adjusting the *type* of Socratic questions asked and the *explicitness* of the guidance provided.

Modulating Guidance for Learner Proficiency

Adaptability is achieved by dynamically shifting the focus of the interaction based on the learner's needs. This involves a progression from concrete to abstract and from explicit to heuristic guidance.

- **For Beginners:** The interaction should be highly concrete. The AI should focus on observable facts (e.g., the text of an error message) and fundamental concepts (e.g., the sequential execution of code). The Socratic questions should probe for basic comprehension ("What does this line of code do?"). The scaffolds offered should be explicit prompts for the simplest, most direct debugging tools, such as using print() statements or reading a traceback. The goal is to build a solid mental model of program execution and introduce foundational debugging practices.
- **For Intermediate Learners:** As students become more comfortable, the AI can shift its focus to more abstract logical reasoning. The questions should challenge their assumptions and prompt them to consider edge cases ("What happens if the input list is empty?"). The scaffolds become more heuristic, suggesting general strategies rather than specific actions ("How could you test that function in isolation?"). This encourages the development of a more systematic and robust testing mindset.
- **For Advanced Learners:** For students with a strong grasp of the fundamentals, the AI's role evolves into that of a high-level design consultant. The Socratic questions can address more abstract and complex topics, such as algorithmic efficiency (e.g., Big O notation), software architecture, design patterns, and the trade-offs between different implementation choices. The guidance becomes metacognitive, prompting the student to reflect on their design decisions ("Why did you choose a list for this task instead of a dictionary? What are the performance implications of that choice?").

The Adaptive Scaffolding Framework Table

The following table provides a practical, operational blueprint for implementing this adaptive scaffolding. It maps learner proficiency levels to their common challenges and prescribes corresponding AI questioning and guidance strategies. This framework allows for the dynamic adjustment of the AI's behavior, ensuring that every student is challenged appropriately within their individual Zone of Proximal Development.

Learner Level	Common Challenges & Cognitive State	AI Questioning Strategy (Socratic Focus)	AI Guidance Strategy (Scaffolding Level)	Example AI Interaction

Beginner	- Misinterpreting syntax errors & tracebacks.	- Flawed mental model of sequential execution. - High frustration; prone to random changes.	**Focus on Concrete Observation & Vocabulary:** "What is the exact text of the error message on line 15?" "Let's trace this by hand. What is the value of x *before* this line runs, and what is its value *after*?"	**Explicit Prompts for Basic Tools:** "A print() statement here would show you the value of the variable in each loop. Would you like to try that?" "Let's read the traceback from the bottom up. The last line usually tells us the specific error."	**Student:** "My code crashes." **AI:** "I see. It can be tricky when that happens. Can you show me the error message it's giving you? Often, the first clue is right there."
Intermediate	- Off-by-one errors in loops. - Incorrect boolean logic in conditionals. - Difficulty with complex data structures (nested lists/dicts). -	**Focus on Hypothesis Testing & Edge Cases:** "What is your assumption about the loop's termination condition?" "What would happen if the input list	**Heuristic Suggestions for Systematic Process:** "How could you test that assumption about the loop? Is there a tool that lets you watch the variable change step-by-ste	**Student:** "My function works for most lists, but not for ``." **AI:** "That's an interesting observation. It suggests the logic might be sensitive to duplicates. What part	

	Overconfidence in assumptions.	contained duplicate numbers?" "Can you think of a scenario where this if statement would fail?"	p?" (Hinting at a debugger). "It might be helpful to isolate this function and test it with a few simple, predictable inputs."	of your code is responsible for handling how many times a number appears?"
Advanced	- Inefficient algorithms (e.g., O(n2) instead of O(n)). - Flawed architectural or design pattern choices. - Subtle state management issues or race conditions. - Underestimating the consequences of a design choice.	**Focus on Abstract Reasoning, Design & Consequences:** "You've used a nested loop. What are the performance implications of that approach as the input size grows?" "Why did you choose this data structure over another, like a hash map?" "What are the	**Metacognitive & Conceptual Prompts:** "Could you explain the trade-offs of your current approach?" "Is there a design pattern that might simplify this state management logic?" "Let's consider an alternative algorithm. How would its complexity compare to your current one?"	**Student:** "This code is too slow on large datasets." **AI:** "Performance can be a challenging puzzle. Let's analyze the algorithm. Where are the most computationally expensive operations in your code? How many times does that inner loop run relative to the size of the input?"

		potential side effects of modifying the list in-place within this function?"		

This adaptive framework represents more than just a guide for tailoring interactions to different students; it outlines a developmental pathway for a single student over time. A truly intelligent tutoring system could learn to recognize a student's growing competence by tracking their responses and the types of errors they make. As the student demonstrates mastery over basic concepts, the system could automatically progress along this spectrum, gradually fading its scaffolds from explicit prompts to more subtle, metacognitive questions. This automated, gradual withdrawal of support is the ultimate embodiment of the Guided Discovery principle, ensuring that the AI tutor evolves alongside the learner it is designed to serve.

Conclusion: Fostering Independent and Resilient Problem-Solvers

The challenge of designing effective AI for education is fundamentally a pedagogical one. An AI assistant that simply provides correct answers may help a student complete an assignment, but it fails at the more important task of teaching them how to think. This report has detailed a robust, theory-driven framework for creating an AI debugging assistant that prioritizes learning over expediency.

Summary of the Socratic Guide Framework

The proposed Socratic Guide Framework, implemented through the detailed master prompt, represents a synthesis of well-established principles from the learning sciences. By combining the disciplined, inquiry-based dialogue of the Socratic method with the structured support of Guided Discovery Learning, this model transforms the AI into a powerful pedagogical agent. Its empathetic tone manages the affective challenges of debugging, while its adaptive scaffolding system ensures that every student is challenged within their Zone of Proximal Development. The AI's strict, core constraint—its refusal to provide direct solutions—is its most important feature, as it preserves the space necessary for genuine student discovery and cognitive engagement.

Beyond Debugging: Cultivating Metacognitive Skills

The ultimate outcome of this approach is not merely a bug-free program, but a more capable and resilient programmer. By consistently engaging in this structured, hypothesis-driven dialogue, students internalize the process of expert debugging. They learn to articulate their intent, systematically isolate problems, form testable hypotheses, and use empirical tools to gather evidence. These are not just coding skills; they are metacognitive, problem-solving skills that are transferable to any complex domain. The AI, in effect, is not just teaching Python; it is teaching a systematic approach to inquiry that is the hallmark of computational thinking and scientific reasoning.

Future Directions and Implications

The Socratic Guide framework presented here has implications that extend far beyond Python debugging. It serves as a blueprint for a new class of AI tutors designed to foster critical thinking in any field that requires complex problem-solving. The core principles—using Socratic questioning to challenge assumptions, providing scaffolds to guide discovery, and adapting the level of guidance to the learner's expertise—are universally applicable. This model could be adapted to guide students through mathematical proofs, the analysis of historical documents, the formulation of scientific hypotheses, or the diagnosis of medical case studies. As AI becomes more integrated into the educational landscape, adopting such pedagogically-grounded frameworks will be essential to ensure that this powerful technology is used not just to automate tasks, but to cultivate the deep thinking, resilience, and intellectual independence that are the true goals of education.