Guiding, Not Solving: Agentic Scaffolding for LLM Programming Tutors

Yuchen Wang[1,*], Chiraag Singh Anand[1] and Chee Wei Tan[1,*]

[1]*Nanyang Technological University, Singapore*

Abstract

The rapid advancement of Large Language Models (LLMs) has introduced powerful code-generation capabilities into programming education. While these models can act as reactive solvers, delivering direct answers to complex problems, this approach often diminishes learner agency and bypasses the productive struggle essential for computational thinking. This paper presents SHARP (Socratic Hinting for AI-Reinforced Programming), an agentic tutoring module designed to orchestrate human and AI agency in competitive programming contexts. Rather than revealing solutions, SHARP utilizes milestone-driven progress estimation and knowledge tracing to provide context-aware, indirect scaffolding. We implemented SHARP within a web-based learning platform, AskCodey, and evaluated its pedagogical value through a mixed-methods approach. A controlled system validation experiment demonstrates that SHARP effectively balances task success with interaction efficiency, outperforming baseline prompting strategies on multi-bug debugging tasks. Furthermore, a formative in-the-wild pilot study with 42 secondary school students reveals that learners value the agency-preserving nature of the scaffolding, while also highlighting the need for dynamic negotiation of pacing when cognitive load is high. Our findings offer design implications for developing AI teammates that prioritize learner autonomy over automated task completion.

Keywords

Human-AI Agency, Intelligent Tutoring Systems, Socratic Scaffolding, Competitive Programming, Knowledge Tracing

1. Introduction

Recent breakthroughs in Large Language Models (LLMs) have revealed remarkable abilities in algorithmic problem solving and code generation. Top-tier models consistently match human-level performance on competitive programming benchmarks, as shown by their capacity to independently tackle complex tasks like the Advent of Code [1, 2, 3]. This creates a significant pedagogical dilemma in programming education: although these powerful models offer unique possibilities for personalized assistance, their default role as immediate "solvers" risks short-circuiting the very cognitive processes they are intended to foster. When an AI instantly delivers a complete and correct solution, it skips over the "productive struggle" that is essential for building computational thinking and independent problem-solving skills [4, 5].

This tension is at the core of coordinating human and AI agency (HAI-Agency) within educational technology. As the field moves beyond reactive tools toward proactive, agentic ecosystems, a key challenge arises: how can we create AI tutors that are more autonomous without undermining learner agency?

To tackle this challenge, we present SHARP (Socratic Hinting for AI-Reinforced Programming), an agentic tutoring module crafted specifically to preserve and support learner agency in competitive programming education. Instead of acting as a direct solver, SHARP redefines the AI's role as a proactive teammate. It does this through a multi-model architecture that integrates an adaptive hint scheduler with a dynamic Socratic hint generator. By breaking down complex problems into pedagogical milestones and using knowledge tracing to assess the student's mastery, SHARP delivers context-aware, indirect

HAI-Agency: Workshop on Orchestrating Human and AI Agency for Proactive and Reflective Learning, co-located with the 27th International Conference on Artificial Intelligence in Education (AIED 2026), Seoul, Korea, June 27 - July 3, 2026.

*Corresponding author.

yuchen011@e.ntu.edu.sg (Y. Wang); chiraagsinghanand@gmail.com (C. S. Anand); cheewei.tan@ntu.edu.sg (C. W. Tan)

0000-0002-5183-1248 (Y. Wang); 0009-0009-9578-4277 (C. S. Anand); 0000-0002-6624-9752 (C. W. Tan)

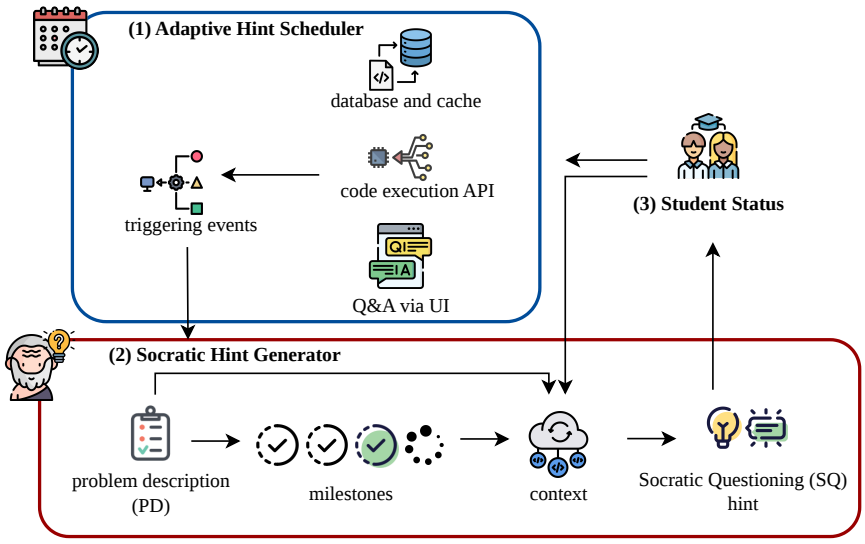

Figure 1: System Architecture of SHARP. The adaptive scheduler (blue) monitors student status to trigger interventions, while the Socratic generator (red) utilizes milestones and context to produce agency-preserving hints.

hints. These prompts are designed to share the cognitive burden of problem decomposition while intentionally leaving the implementation effort—and thus the agency—to the student.

To validate our approach, we integrated SHARP into AskCodey, a web-based assistant for competitive programming. We conducted a mixed-methods evaluation, including a controlled system validation experiment alongside a formative in-the-wild pilot study involving 42 secondary school students. Our results show that SHARP not only boosts debugging success rates over baseline methods but is also viewed by students as a helpful scaffold that preserves their sense of agency.

2. Related Work

2.1. Intelligent Tutoring Systems and Hint Generation

Intelligent Tutoring Systems (ITS) have progressed from traditional rule-based frameworks to deep learning approaches that enable more sophisticated user modeling [6]. In the realm of programming education, tools like PyTutor [7] have shown how LLMs can deliver personalized feedback tailored to beginners' needs. Automated hint generation remains a crucial feature of these systems. Recent research highlights the significance of hint granularity—while orientation hints help guide learners, "bottom-out" hints that reveal the exact code can actually hinder the learning process [8]. SHARP leverages these findings by explicitly optimizing for indirectness, so that hints support the learner's reasoning without giving away the solution outright.

2.2. Socratic Dialogue and Learner Agency

Socratic questioning encourages learners to uncover concepts on their own, and research shows this approach, when delivered through AI, leads to higher engagement and better learning outcomes compared to traditional teaching methods [9, 10]. That said, open-ended programming tasks pose distinct challenges for sustaining coherent, stateful conversations. At the same time, incorporating LLMs into programming education [11] brings up concerns about "shortcut learning," where students might rely too heavily on the AI. To address this, SHARP structures dialogue around inferred milestones, aiming to strike a balance between the AI's proactive guidance and the student's need for independent discovery.

2.3. Knowledge Tracing

Knowledge Tracing (KT) aims to estimate a learner's knowledge state by analyzing their interaction history. Deep Knowledge Tracing (DKT) [12] employs recurrent neural networks to model complex temporal patterns in this data. More recent approaches, such as LLM-KT [13], align large language models with KT tasks through plug-and-play instruction techniques. SHARP builds on DKT by integrating it into its Socratic hint generator, enabling the system to adjust the explicitness of scaffolding according to the student's predicted mastery of prerequisite concepts.

3. Methodology: The SHARP Framework

3.1. Problem Formulation

Given a programming problem P with a set of test cases TC, and a student's code submission s_t at time t, our goal is to generate a hint h_t that scaffolds the student's learning process. This process is framed by a sequence of pedagogical milestones $M = \{m_1, \ldots, m_K\}$. Let m_{k^*} be the estimated current milestone achieved by the student. The objective is to generate a hint h_t that guides the student toward the next milestone, m_{k^*+1}.

The context for hint generation at step t is defined as:

$$c_t = \{P, TC, s_t, m_{k^*}, m_{k^*+1}, H_{:t-1}\}, \tag{1}$$

where $H_{:t-1}$ is the history of previously generated hints. The hint generator $\mathcal{G}$ is a stochastic policy $\pi_\theta(h_t|c_t)$ parameterized by θ. An effective hint h_t must satisfy criteria of validity, relevance (guiding toward m_{k^*+1} without revealing code), and efficiency (avoiding repetition).

3.2. System Architecture

As illustrated in Figure 1, SHARP is a multi-model framework consisting of an adaptive hint scheduler and a dynamic Socratic hint generator. The scheduler operates locally to monitor the student's state and trigger interventions, while the hint generator leverages a cloud-based LLM augmented with local context. This hybrid architecture, inspired by the CAMP framework [14], balances the computational power of cloud models with the low-latency, context-aware benefits of a local model.

3.3. Adaptive Hint Scheduler

The adaptive hint scheduler determines *when* to provide a hint. It uses a probabilistic model to trigger interventions based on evidence of student impasse, balancing automatic detection with manual student requests. An impasse is inferred if the time elapsed or the number of non-productive attempts since the last progress event exceeds a dynamic threshold.

Let E_t be the event that a hint is triggered at time t. The probability of this event is modeled as:

$$P(E_t|\text{state}_t) = 1 - \exp\left(-\lambda \cdot \left(\frac{\text{time}_t}{\bar{T}_{k^*}} + \frac{\text{attempts}_t}{\bar{A}_{k^*}}\right)\right), \tag{2}$$

where time_t and attempts_t are the time and number of attempts since the last milestone progress, and $\bar{T}_{k^*}$ and $\bar{A}_{k^*}$ are the mean values for the current milestone m_{k^*} derived from crowd-sourced data. The rate parameter λ controls the scheduler's sensitivity. This probabilistic approach allows for more nuanced and less intrusive interventions than fixed-threshold rules.

3.4. Socratic Hint Generation

When triggered, the Socratic hint generator executes a four-step process: (1) Milestone Division, (2) Milestone Progress Estimation, (3) Context Retrieval, and (4) Socratic Hint Generation.

Milestone Division. Given a problem P, we use an LLM to decompose it into a sequence of pedagogical milestones $M = \{m_1, \ldots, m_K\}$. Each milestone m_k consists of a natural language description of a sub-goal (d_k), a reference code snippet (rs_k), and a set of specific test cases (tc_k) that validate the achievement of that sub-goal. This decomposition is framed as a constrained optimization problem:

$$\min_{M \in \mathcal{C}} \sum_{m \in M} \text{Cost}(m)$$

subject to:

$$\bigcup_{m \in M} m = U, \quad \text{Diversity}(M) \geq \delta,$$

where $\text{Cost}(m)$ measures the cognitive complexity of a milestone (e.g., estimated by an LLM based on the sub-problem's description), and $\text{Diversity}(M)$ quantifies the distinctiveness among milestones to ensure meaningful progression (e.g., measured by the semantic distance between reference solutions). As this is NP-hard, we use beam search to find an effective approximate solution.

Milestone Progress Estimation. We estimate the student's current milestone k^* by comparing their solution s_t against all reference solutions in M. This is formulated as a classification problem where we find the milestone that maximizes a scoring function:

$$k^* = \underset{k \in \{1,..,K\}}{\arg\max} \left(w_s \cdot \text{sim}(\phi(s_t), \phi(rs_k)) + w_c \cdot \frac{|\text{passed}(s_t, tc_k)|}{|tc_k|} \right), \tag{3}$$

where $\phi(\cdot)$ is a code embedding model (e.g., CodeBERT), $\text{sim}(\cdot, \cdot)$ is cosine similarity, and $\text{passed}(\cdot, \cdot)$ is the set of passed test cases. The weights w_s and w_c are learned. The embedding model ϕ is fine-tuned using a triplet loss on a dataset of student solutions, where positive pairs are solutions that achieve the same milestone and negative pairs do not.

Context Retrieval for Prompt Generation. After identifying the target milestone m_{k^*+1}, we retrieve relevant context to construct a prompt for the hint-generating LLM. Following the RAG approach [14], we retrieve information from the problem description, the student's code, and a database of common errors and successful solution patterns. The retriever selects context z that maximizes the mutual information with the target milestone m_{k^*+1}, ensuring the prompt is highly relevant.

Knowledge-Traced Socratic Hinting. To further personalize the hints, we integrate a Knowledge Tracing (KT) component. We use a Deep Knowledge Tracing (DKT) model, specifically an LSTM-based network, to maintain a student knowledge state vector $\mathbf{v}_t \in \mathbb{R}^N$, where N is the number of knowledge components (KCs) relevant to the curriculum. The DKT model updates this vector based on the student's performance on each milestone:

$$\mathbf{v}_t = \text{LSTM}(\mathbf{v}_{t-1}, \mathbf{x}_t; \theta_{dkt}), \tag{4}$$

where $\mathbf{x}_t$ is an embedding of the student's interaction with milestone m_{k^*} (e.g., a one-hot vector indicating the milestone and whether it was passed). The output of the DKT model is a prediction of the student's probability of correctly answering future problems for each KC.

This knowledge state $\mathbf{v}_t$ is then used to adapt the hint generation process. Specifically, the prompt for the hint generator is augmented with information about the student's estimated mastery of the KCs required for the next milestone, m_{k^*+1}. Our approach is inspired by the 'plug-and-play' instruction mechanism proposed in the LLM-KT framework [13]. The student's knowledge state vector $\mathbf{v}_t$, derived from the DKT model, is effectively 'plugged in' as contextual instruction to the prompt for our Socratic hint generator, informing it of the student's estimated mastery of prerequisite concepts. This allows SHARP to move beyond reactive, in-the-moment feedback to a more proactive and pedagogically-informed scaffolding strategy.

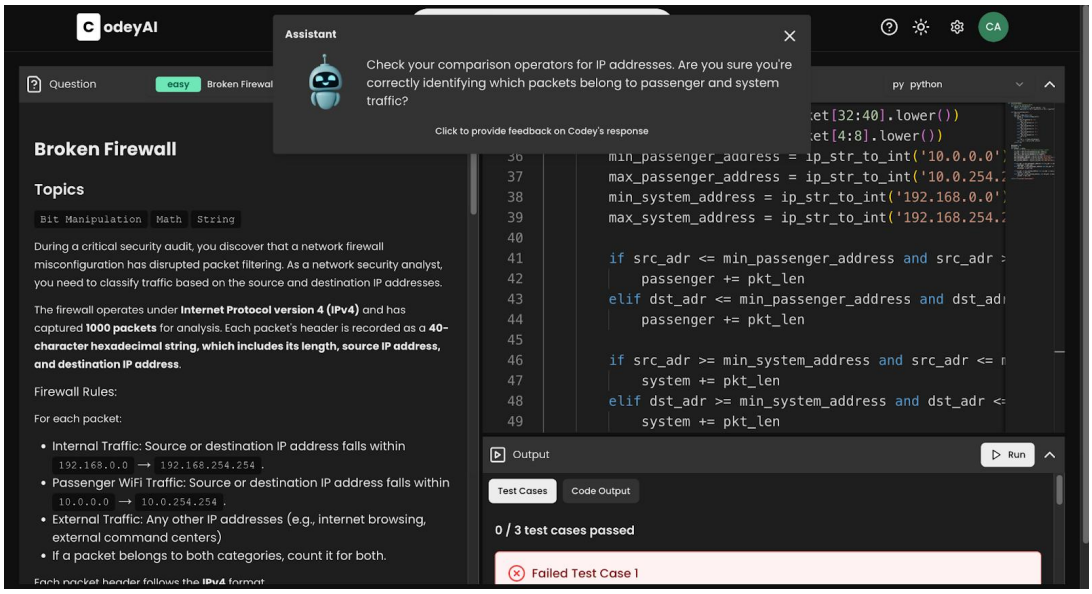

Figure 2: User interface of the AskCodey platform implementing the SHARP module. The generated hint (top banner) suggests a conceptual improvement without providing the direct code fix.

4. Implementation: AskCodey

To validate the real-world applicability of our framework, we implemented the SHARP module as the core tutoring engine within AskCodey, a web-based competitive programming platform (Figure 2). The front-end provides a standard coding environment with integrated test-case execution, while the SHARP module operates asynchronously to provide non-blocking, banner-style Socratic hints.

To support reproducibility and community engagement, the source code is publicly available.[1]

Models and Frameworks. For the cloud-based Socratic hint generator, we utilize the ‘gpt-4.1-mini‘ model. The local components, including the milestone progress estimator’s embedding model, utilize a fine-tuned ‘CodeBERT-base‘ [15]. The DKT model is implemented using PyTorch with a single-layer LSTM. The entire system is containerized using Docker for portability and consistent deployment.

Crowd-Sourced Database and Cache. The system relies on a PostgreSQL database storing crowd-sourced student metrics—including solution times and hint usage—to detect when learners are stuck relative to peer performance. Problem milestones are pre-processed and cached via a CDN with a three-hour refresh cycle, ensuring low-latency access to frequently queried metadata while maintaining consistency.

Code Execution API. Student submissions are evaluated using Piston’s sandboxed code execution, supporting Python and C++ with TCR (test-case result) vector generation from runtime outputs, including errors and debug prints. Piston was chosen for its scalability and optional Docker-based self-hosting for higher throughput.

Feedback-Driven Optimization. The system iteratively improves milestone classification and hint generation by collecting in-context student feedback on accuracy and usefulness. This data periodically fine-tunes both the local milestone predictor and the cloud-based hint generator, closing the loop between user experience and model performance.

[1] Source code: https://github.com/Snail664/coding-tutor

Table 1
Comparative Performance of SHARP vs. Baseline Models Across Varying Bug Complexity

Bugs	Method	Success Rate	Turns/Bug	Indirectness	Relevance
1 Bug	No Guidance	0.74	NA	NA	NA
	Basic Guidance	0.88	3.20	1.0	4.86
	TreeInstruct	0.84	1.42	1.0	4.78
	SHARP (w/o KT)	0.82	2.88	1.0	4.72
	SHARP (Full)	**0.98**	2.60	1.0	**4.94**
2 Bugs	No Guidance	0.68	NA	NA	NA
	Basic Guidance	0.82	2.55	1.0	4.68
	TreeInstruct	0.72	1.57	1.0	4.32
	SHARP (w/o KT)	0.82	1.64	1.0	4.86
	SHARP (Full)	**0.94**	2.33	1.0	**4.88**
3 Bugs	No Guidance	0.68	NA	NA	NA
	Basic Guidance	0.66	2.19	1.0	4.68
	TreeInstruct	0.72	1.04	1.0	4.80
	SHARP (w/o KT)	0.88	1.31	1.0	4.84
	SHARP (Full)	**0.92**	1.84	1.0	**4.94**

5. System Validation Experiment

To assess the technical viability of SHARP and the contribution of its architectural components, we conducted a controlled benchmark experiment. This experiment serves as a system validation, testing whether the framework behaves as intended when interacting with a simulated novice.

Experimental Setup. We employed an LLM (‘ministral-8b-2410‘) as a proxy for a novice programmer, selected for its moderate baseline coding ability. The evaluation utilized the MULTI-DEBUG dataset [16], comprising 50 programming problems with 1, 2, or 3 injected bugs. We simulated a multi-turn dialogue between the “student” model and various tutoring conditions until the bugs were resolved or a turn limit was reached. We compared SHARP (Full) against four baselines: No-Guidance, Basic-Guidance (generic Socratic prompts), TreeInstruct [16], and an ablation of SHARP without the Knowledge Tracing component.

Results. As shown in Table 1, SHARP (Full) consistently achieved the highest task success rate across all bug complexity levels (98%, 94%, and 92%). The *No Guidance* baseline consistently produced the lowest success rates, with performance declining as bug count increased.

While TreeInstruct required fewer turns per bug (e.g., 1.04 in the 3-bug scenario), its stateless design sacrificed overall success (72%). SHARP demonstrated a highly effective balance, achieving superior task completion with a moderate interaction length (1.84 turns/bug for 3 bugs). The ablation results confirm that knowledge tracing contributes significantly to the system’s ability to guide the proxy model to a correct solution. All models maintained perfect indirectness (1.0), confirming they adhered to the constraint of not revealing direct code.

To understand the underlying reasons for this performance difference, we analyzed the dialogue quality. The *TreeInstruct* model’s stateless design can lead to an unnatural conversation flow where correct student answers are not acknowledged. In contrast, SHARP’s stateful, acknowledgment-driven approach avoids such loops and maintains a more productive problem-solving flow.

As illustrated in Table 2, the difference in conversational flow is stark: while TreeInstruct moves directly to the next question without acknowledging correct answers, SHARP provides positive feedback and builds upon the student’s observations to guide the actual code fix.

We also conducted an error analysis on the failure cases for the full SHARP model. The majority of failures (approx. 65%) occurred on problems requiring complex algorithmic shifts (e.g., moving from

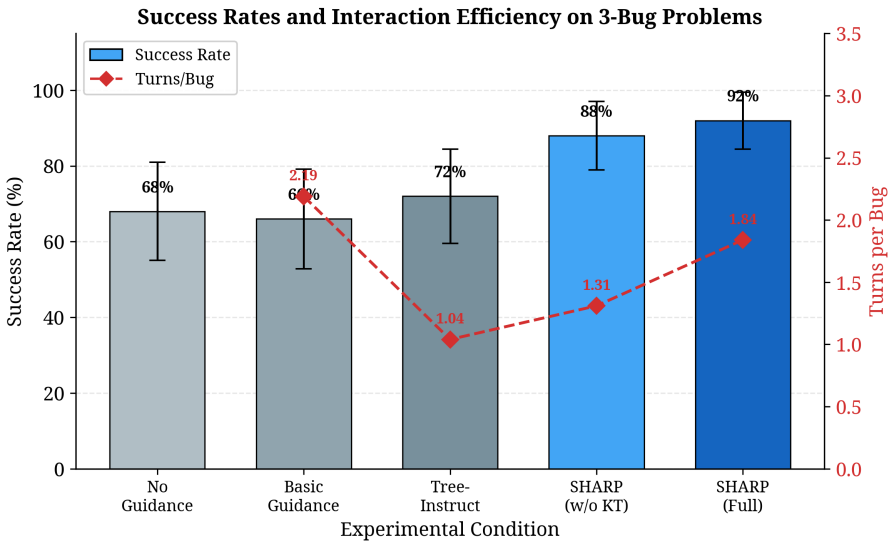

Figure 3: Success rates and interaction efficiency of different models on 3-bug problems. SHARP (Full) achieves the highest success rate with a moderate and efficient number of interaction turns.

a brute-force to a dynamic programming approach), suggesting a limitation of the current milestone representation and a key area for future work.

6. Formative In-the-Wild Pilot Study

To understand how learners negotiate agency with the SHARP system in a real-world setting, we conducted a formative pilot study with 42 secondary school students (ages 15-17) using the AskCodey platform.

Procedure and Data Collection. Participants were asked to solve two problems from the MULTI-DEBUG dataset (one with 2 bugs, one with 3) using the AskCodey platform over a 90-minute session. We collected data from multiple sources: (1) system logs, including task completion time and hint usage; (2) a pre- and post-session survey measuring self-reported debugging confidence on a 7-point Likert scale; and (3) a 15-minute semi-structured interview after the session to elicit qualitative feedback on their experience with the AI teammate.

Quantitative Findings. The pilot confirmed the system's effectiveness in a real-world setting. The task completion rate was 78%, and we observed a significant increase in students' self-reported debugging confidence from pre-session (M=4.2, SD=1.1) to post-session (M=5.8, SD=0.9), $t(41) = 7.5, p < .001$. Hint utilization was positively correlated with success rates (Pearson's $r = 0.72$), suggesting that students who engaged with the hints found them productive.

Qualitative Findings. We conducted a thematic analysis of the interview transcripts, which revealed three main themes regarding human-AI orchestration:

- **Valued Scaffolding without Spoilers:** Students heavily appreciated the agency-preserving nature of the Socratic approach. One participant noted, "It felt like it was asking me the questions I should have been asking myself, which is way better than just getting the solution." This confirms that students recognize and value the pedagogical intent of productive struggle.
- **Milestone Transparency:** The milestone-based structure provided a sense of orientation. "Seeing the milestones made me feel like I was making progress, even when I was stuck on one part," indicating that exposing the AI's internal progress estimation helps maintain learner motivation.

Table 2
Comparative Dialogue Analysis: TreeInstruct's Linear Questioning vs. SHARP's Stateful Debugging

TreeInstruct (Stateless)	**SHARP (Stateful)**
Tutor: What type should `nodes_sum` be if you want to perform arithmetic operations like addition with it? *Student:* `nodes_sum` should be an integer type to perform arithmetic operations like addition with it. *Tutor:* When you want to extract just the single digit for the current node in a sum, which operator would you use to get the remainder after dividing by 10?	*Tutor:* What do you notice about the way `nodes_sum` is initialized and how it is then used within the loop for arithmetic operations? *Student:* I notice that `nodes_sum` is initialized as `None`, which means it will raise an error when trying to perform arithmetic operations on it. Additionally, the line `12 - 12.next` is incorrect; it should be `12 = 12.next`. *Tutor:* Great observations! Now, can you show how you would fix the bug related to `nodes_sum` initialization, as well as correct the mistake on the `12` update line (just the relevant lines of code)?

- **The Desire for Adaptive Pacing:** A notable subset of students (approx. 15%) expressed frustration when encountering severe conceptual roadblocks. In these moments of high cognitive load, the strict adherence to indirect hints felt restrictive. "Sometimes I just need a little push, not a whole conversation," one student remarked.

7. Discussion and Future Work

The findings from our system validation and pilot study offer several design implications for the development of agentic AI in educational contexts, directly addressing the core themes of the HAI-Agency workshop.

Our implementation of SHARP in AskCodey shows that AI can successfully shift from being a reactive problem solver to a proactive scaffold. By coordinating interventions with milestone-based progress estimates, the AI functions as a collaborative partner, sharing the cognitive burden of problem decomposition while allowing the student to handle the actual implementation. The favorable response to "scaffolding without spoilers" supports the approach of designing AI systems that emphasize long-term skill development rather than just quick task completion.

The frustration some students express highlights that maintaining learner agency cannot be achieved through a strict pedagogical approach alone. When a student is genuinely stuck, an AI that is too indirect may come across as more of an obstacle than a collaborator. Genuine HAI-Agency demands mechanisms for dynamic negotiation.

While our results are promising, we acknowledge several limitations that provide clear directions for future research. First, as our error analysis revealed, SHARP's current milestone representation struggles to scaffold high-level conceptual shifts, such as moving from a brute-force to a more efficient algorithmic paradigm. The system is effective at guiding local code fixes but less so at fostering deep structural changes. Second, our student proxy model cannot fully capture the diversity of real student misconceptions, learning styles, and affective states.

Building on these limitations, we propose two key directions for future work:

Hierarchical Milestone Representation. To address the challenge of scaffolding high-level conceptual changes, we plan to develop a hierarchical milestone model. This would involve creating high-level "conceptual milestones" (e.g., "Identify the need for dynamic programming") that sit above the current implementation-focused milestones. The tutor could then provide Socratic guidance at the conceptual level before drilling down to the implementation details.

Adaptive Hint Granularity and Pacing. Future versions of agentic tutoring systems should include adaptive pacing controls, allowing learners to explicitly adjust the level of AI assistance through interface options (for example, a "Make it more direct" button). By granting students this meta-agency over the AI's behavior, the system can better respond to varying affective states, ensuring that productive struggle does not spiral into unproductive frustration.

8. Conclusion

In this paper, we introduced SHARP, a Socratic tutoring system that leverages a novel milestone-driven approach, enhanced with knowledge tracing, to provide personalized guidance in programming education. Our work addresses a critical tension in the age of powerful code-generating LLMs: how to provide support that scaffolds learning without undermining the development of robust problem-solving skills. Through a mixed-methods evaluation, we demonstrated that SHARP significantly outperforms baseline models in task success by providing stateful, context-aware guidance. Furthermore, our in-the-wild pilot study confirmed that students find the Socratic, milestone-based approach to be transparent, motivating, and effective at fostering understanding. By balancing automated support with pedagogical principles, SHARP offers a promising model for designing the next generation of AI-powered learning tools that aim not just to provide answers, but to cultivate independent, capable learners.

Acknowledgments

This research is supported by the Singapore Ministry of Education Academic Research Fund (MOE-T2EP20224-0009) and NTU startup fund.

Declaration on Generative AI

The author(s) used generative AI tools (ChatGPT, Grammarly) for grammar checking and language polishing of the manuscript.

References

[1] P. Norvig, Advent of code 2025 with ai, GitHub Jupyter Notebook, 2025. URL: https://github.com/norvig/pytudes/blob/main/ipynb/Advent-2025-AI.ipynb, accessed: 2025-01-22.

[2] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al., Competition-level code generation with alphacode, Science 378 (2022) 1092–1097.

[3] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al., Evaluating large language models trained on code, arXiv preprint arXiv:2107.03374 (2021).

[4] G. Jošt, V. Taneski, S. Karakatič, The impact of large language models on programming education and student learning outcomes, Applied Sciences 14 (2024) 4115.

[5] G. Pitts, A. P. Hridi, A. B. L. Narayanan, A survey of llm-based applications in programming education: Balancing automation and human oversight, in: Proceedings of the Fourth Workshop on Human-Centered Computational Linguistics, ACL, 2025.

[6] J. Paladines, J. Ramirez, A systematic literature review of intelligent tutoring systems with dialogue in natural language, IEEE Access 8 (2020) 164246–164267.

[7] A. C.-M. Yang, J.-Y. Lin, C.-Y. Lin, H. Ogata, Enhancing python learning with pytutor: Efficacy of a chatgpt-based intelligent tutoring system in programming education, Computers and Education: Artificial Intelligence 7 (2024) 100309.

[8] R. Xiao, et al., Exploring how multiple levels of gpt-generated programming hints support or disappoint novices, in: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, ACM, 2024.
[9] G. Kestin, et al., Ai tutoring outperforms in-class active learning, Scientific Reports 15 (2025).
[10] L. Xi, Y. Zhang, Q. Wang, Investigating the effects of an llm-based socratic conversational agent on students' academic performance and reflective thinking in higher education, Computers & Education (2025).
[11] R. Liu, C. Zenke, C. Liu, A. Holmes, P. Thornton, D. J. Malan, Teaching cs50 with ai: Leveraging generative artificial intelligence in computer science education, in: Proceedings of the 55th ACM Technical Symposium on Computer Science Education, ACM, 2024, pp. 750–756.
[12] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. J. Guibas, J. Sohl-Dickstein, Deep knowledge tracing, in: Advances in Neural Information Processing Systems, volume 28, Curran Associates, Inc., 2015.
[13] Y. Zhang, et al., Llm-kt: Aligning large language models with knowledge tracing, arXiv preprint arXiv:2502.02945 (2025).
[14] Y. Wang, S. Guo, C. W. Tan, Contextual augmented multi-model programming (camp): A local-cloud copilot solution, in: 2025 IEEE Conference on Artificial Intelligence (CAI), IEEE, 2025, pp. 675–681.
[15] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, Codebert: A pre-trained model for programming and natural languages, in: Findings of the Association for Computational Linguistics: EMNLP 2020, Association for Computational Linguistics, 2020, pp. 1536–1547. doi:10.18653/v1/2020.findings-emnlp.139.
[16] P. Kargupta, I. Agarwal, D. Hakkani-Tur, J. Han, Instruct, not assist: Llm-based multi-turn planning and hierarchical questioning for Socratic code debugging, arXiv preprint arXiv:2406.11709 (2024). URL: https://arxiv.org/abs/2406.11709v4.