

SHARP: Socratic Hinting for AI-Reinforced Programming in Competitive Programming Education

Yuchen Wang^{1,*}, Chiraag Singh Anand¹ and Chee Wei Tan^{1,*}

¹Nanyang Technological University, Singapore

Abstract

Recent advances in Large Language Models (LLMs) have demonstrated strong capabilities in generating correct and complex code, raising a critical pedagogical challenge: how to effectively use these models to teach computational thinking without fostering student dependency on automated solutions. We present SHARP, a generative AI-driven tutoring system for competitive programming that utilizes a milestone-based architecture to provide Socratic-inspired hints. SHARP fosters “productive struggle” by avoiding direct solution delivery in favor of context-aware, incremental guidance through an adaptive hint scheduler and a dynamic Socratic hint generator. We evaluate SHARP through a controlled benchmark using an LLM proxy across 50 problems with varying bug complexity, demonstrating that SHARP generates highly relevant, indirect hints that guide progress more effectively than baseline methods. An ablation study confirms the critical contribution of the milestone estimation component. Furthermore, an exploratory classroom deployment with 42 secondary-school students provides preliminary evidence of the system’s usability and a positive correlation between hint usage and debugging confidence. We discuss the implications of integrating real-time execution feedback with generative AI for personalized, pedagogically sound support in computer science education.

Keywords

Intelligent Tutoring Systems, Large Language Models, Socratic Questioning, Milestone-Based Scaffolding, Competitive Programming

1. Introduction

Recent advances in large language models (LLMs) have demonstrated strong capabilities in competitive programming, a domain once thought to require uniquely human ingenuity [1, 2, 3]. The startling proficiency of modern LLMs was recently underscored by Peter Norvig’s 2025 experiment, where leading models successfully solved all 25 days of the notoriously difficult Advent of Code challenge [4]. This rapid progress presents a *pedagogical paradox*: if AI can generate correct and complex code on demand, how do we effectively teach the next generation of programmers to think computationally and solve problems independently?

While LLMs offer unique possibilities for personalized assistance—from interactive copilots [5] to alignment with human feedback [6]—their default role as immediate solvers risks short-circuiting cognitive processes [7]. In secondary education, where strong programming foundations must be built early [8], simply providing solutions fosters dependency and undermines core problem-solving skills [9]. Existing models often encourage “solution snarfing” behaviors, causing “cognitive debt” [10]—while AI assistance improves accessibility, it may inadvertently hinder computational thinking [11].

We present SHARP, a competitive programming tutoring system that encourages independent thinking through a Socratic-inspired hinting framework. As illustrated by Figure 1, SHARP implements a two-phase architecture: (1) an *adaptive hint scheduler* that mines crowd-sourced interaction data—including timing patterns and attempt counts across the student cohort—to determine optimal intervention points, and (2) a *dynamic hinting engine* that generates Socratic prompts based on learned student progress and retrieved contextual information. Student code submissions are executed in a sandboxed environment, test-case results inform automated progress assessment, and collected feedback continuously refines

10th Educational Data Mining in Computer Science Education (CSEDM) Workshop, June 28, 2026, Seoul, Republic of Korea

*Corresponding author.

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

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



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

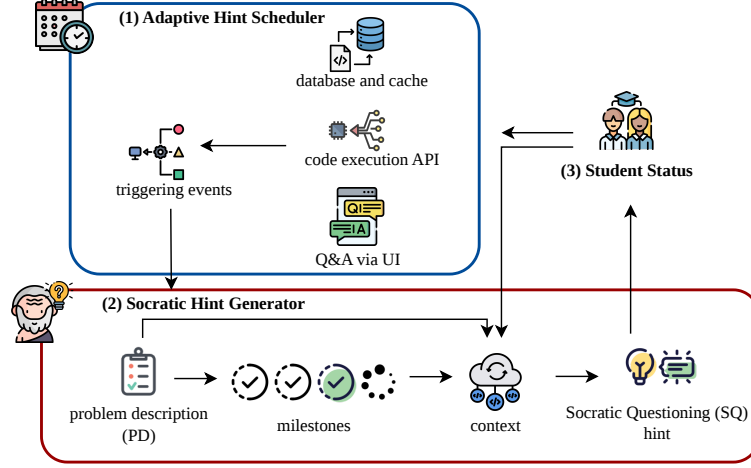


Figure 1: Architecture of SHARP. (1) The Adaptive Hint Scheduler monitors student states, triggering (2) the Socratic Hint Generator to produce guided Socratic questions that support problem-solving. (3) Student progress updates the local database, creating a data-driven feedback loop for AI-assisted programming education.

the hint generation pipeline. Unlike existing AI-assisted programming tools that focus on correctness and completeness, SHARP employs a step-by-step milestone checking approach to gradually guide the student toward the next goal while maintaining sufficient space for independent exploration.

2. Related Work

2.1. Automated Hint Generation and ITS

Intelligent Tutoring Systems (ITS) have a long history of providing customized instruction in STEM fields [12, 13]. Prior to LLMs, extensive research explored automated feedback in programming education: Keuning et al. [14] highlighted challenges in providing adaptable, context-aware guidance; Crow et al. [15] noted early systems’ reliance on rigid, domain-specific models; and McBroom et al. [16] synthesized these efforts, emphasizing the importance of timing and indirectness. The recent integration of LLMs has enabled new ITS capabilities: PyTutor [17] provides personalized Python feedback, and Fodouop Kouam et al. [18] found that adaptive systems significantly improve outcomes for novices. However, Jošt et al. [19] found that informal LLM usage may negatively affect independent problem-solving ability [2, 20], underscoring the need for pedagogical frameworks that balance automation with learning.

2.2. Socratic Dialogue for Programming

Socratic questioning guides learners toward discovering concepts independently [21]. Kestin et al. [22] found that AI Socratic tutoring outperformed traditional active learning, and Xi et al. [23] reported positive effects on reflective thinking. For programming specifically, Roest et al. [24] explored LLM-generated next-step hints, while Phung et al. [25] proposed a two-model approach for hint generation and validation. Studies indicate that orientation hints are effective but bottom-out hints risk undermining learning [26]. SHARP focuses on generating indirect, milestone-driven hints to maintain pedagogical value.

2.3. AI-Assisted Competitive Programming Education

Competitive programming presents unique challenges: problems require multi-step algorithmic reasoning, debugging involves tracing complex logic, and students must develop independent strategies

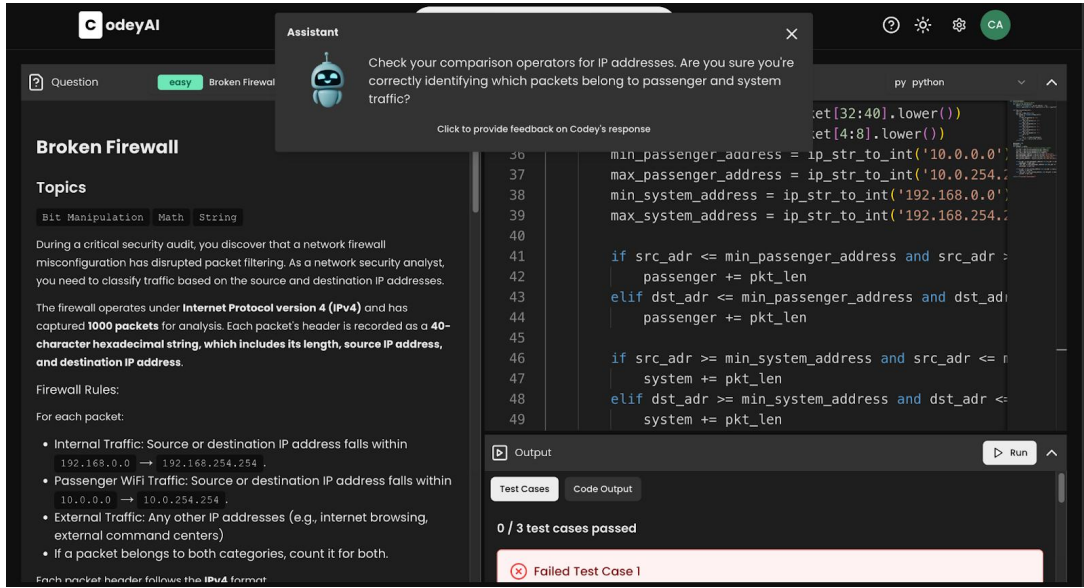


Figure 2: Ask Codey User Interface. The system provides contextual, Socratic-inspired hints (e.g., prompting the user to check comparison operators for IP addresses) rather than directly revealing the solution code.

under time pressure. Prior tools such as iSnap [27] and Nemobot [28] enriched interactive programming for novices, while AutoTutor [29] demonstrated how natural language dialogue fosters deeper understanding. SHARP extends these ideas by integrating real-time code execution analysis with milestone-based progress estimation, providing context-aware Socratic feedback tailored to multi-step debugging challenges.

3. Problem Formulation

Given a programming problem P with test cases TC and a student's solution s_t at step t , we aim to (1) identify the student's current milestone m_{k^*} in the solution sequence $M = \{m_1, \dots, m_M\}$ (where m_1 is empty and m_M is complete), and (2) generate a hint h_t that maximizes independent progress toward m_{k^*+1} . The coding context at step t is defined as:

$$c_t = \{P, TC, s_{t-1}, m_{k^*}, m_{k^*+1}\},$$

where k^* is the estimated milestone index.

The hint generator $\mathcal{G}$ produces $h_t = \mathcal{G}(h_{t-1}, c_t)$, where h_{t-1} denotes the hint history. An effective hint h_t must satisfy three criteria: (i) *Validity*: grammatically correct and logically sound, (ii) *Relevance*: guides toward m_{k^*+1} via Socratic questioning without revealing the solution directly, and (iii) *Efficiency*: provides novel insight compared to previous hints.

4. System Design and Methodology

4.1. Adaptive Hint Scheduler

The adaptive hint scheduler determines when a hint should be provided, serving as the *heartbeat* of the system. Scheduling decisions consider two scenarios: *detected student impasse* and *manual trigger*. A hint is issued when the student is inferred to be stuck or when the student explicitly requests assistance.

Automatic impasse detection applies educational data mining techniques to crowd-sourced interaction logs. By aggregating timing and attempt data across the student cohort, the system builds a statistical model of expected behavior at each milestone, identifying when an individual student's trajectory

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
	TreelInstruct	0.84	1.42	1.0	4.78
	SHARP (w.o. Milestone est.)	0.82	2.88	1.0	4.72
	SHARP (w. Milestone est.)	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
	TreelInstruct	0.72	1.57	1.0	4.32
	SHARP (w.o. Milestone est.)	0.82	1.64	1.0	4.86
	SHARP (w. Milestone est.)	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
	TreelInstruct	0.72	1.04	1.0	4.80
	SHARP (w.o. Milestone est.)	0.88	1.31	1.0	4.84
	SHARP (w. Milestone est.)	0.92	1.84	1.0	4.94

deviates from the norm. Let t denote the elapsed time since the last progress event; the trigger condition is:

$$t \geq \max(\mathbb{E}_s[t_s], \tau),$$

where $\mathbb{E}_s[t_s]$ is the mean time spent by all students at the same milestone, and τ is a fixed threshold for cold-start scenarios. The system also triggers a hint if the student exceeds a predefined number δ of incorrect attempts.

4.2. Socratic Hint Generation Pipeline

When triggered, the generator executes a four-step process:

1. Milestone Division. An LLM decomposes problem P into a sequence of pedagogical milestones $M = \{m_1, \dots, m_K\}$. Each m_k consists of a natural language description, a reference code snippet, and specific test cases. This is framed as a constrained optimization problem minimizing $\sum \text{Cost}(m)$ subject to full coverage and $\text{Diversity}(M) \geq \delta$. We use beam search to find an approximate solution.

2. Milestone Progress Estimation. We estimate the student’s current milestone k^* by comparing their solution s_t against reference solutions, combining code semantics with functional test results:

$$k^* = \arg \max_{k \in \{1, \dots, K\}} \left(w_s \cdot \text{sim}(\phi(s_t), \phi(rs_k)) + w_c \cdot \frac{|\text{passed}(s_t, tc_k)|}{|tc_k|} \right), \quad (1)$$

where $\phi(\cdot)$ is a CodeBERT [30] embedding fine-tuned using triplet loss, rs_k is the reference solution for milestone k , and tc_k denotes the test cases associated with that milestone. The weights w_s and w_c balance semantic similarity against functional correctness.

3. Context Retrieval. Following the RAG approach [31], we retrieve information from the problem description, student’s code, and a database of common errors. The retriever selects context z that maximizes mutual information with target milestone m_{k^*+1} .

4. Socratic Hint Generation. The retrieved context is used to construct prompts for context-aware Socratic hint generation. The generator considers the input x , the retrieved context z , and the history of previous prompts to produce a hint that guides the student toward the next milestone without revealing the solution directly.

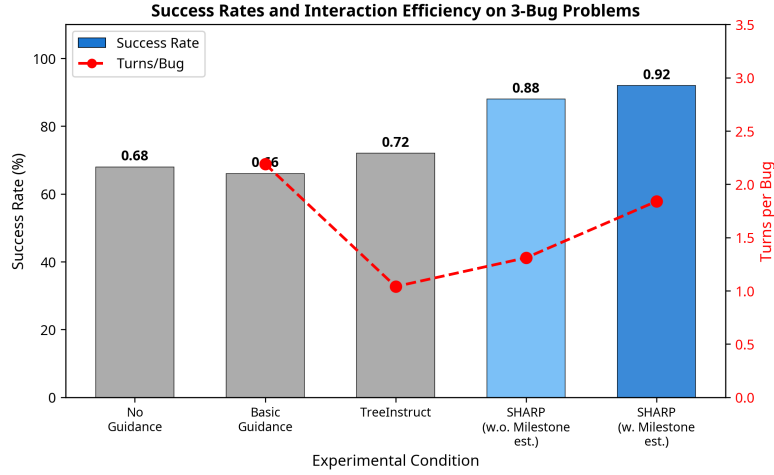


Figure 3: Success rates on 3-bug problems. SHARP (w. Milestone est.) outperforms all baselines across all conditions.

4.3. Implementation Details

We integrated SHARP into Ask Codey¹, a web-based competitive programming assistant deployed at an international secondary school (Figure 2). The platform is built on Next.js (React) with PostgreSQL via Prisma ORM, Auth0 authentication, and Upstash Redis for rate limiting. Student submissions are evaluated using Piston’s sandboxed code execution API, supporting Python and C++. A language-specific test harness runs all test cases, captures outputs, and generates a structured test-case result (TCR) vector that informs milestone progress estimation.

The core hint generation relies on Claude 3.5 Sonnet as the primary LLM, with GPT-4o as an automatic fallback. System prompts enforce strict constraints: hints must be concise, focus on one immediate error, and avoid revealing solution code. Text hints are simultaneously converted to speech via OpenAI’s TTS-1 model. The system also collects in-context student feedback—storing prompts, code state, LLM output, and user ratings—creating a closed loop for continuous refinement.

5. Evaluation

We designed a mixed-methods evaluation guided by three research questions: **RQ1:** How does SHARP compare to baseline tutoring strategies in terms of task success rate, interaction efficiency, and pedagogical quality? **RQ2:** What is the contribution of the milestone estimation component? **RQ3:** How do students perceive the utility and quality of SHARP’s guidance in a real-world setting?

5.1. Technical Validation via LLM Proxy Benchmark

Evaluating adaptive tutoring systems is challenging because human learning dynamics are complex and variable. As a technical validation step, we utilized an LLM (ministral-8b-instruct-2410) as a proxy for a novice student. While an LLM cannot fully capture human affective states, it provides a deterministic, reproducible environment to evaluate whether generated hints are logically sound, relevant, and capable of guiding flawed code toward a correct solution.

Dataset. We used the MULTI-DEBUG dataset [11], comprising 50 programming problems from competitive programming platforms with 1, 2, or 3 injected bugs designed to emulate common novice errors. We curated executable test cases from LeetCode for automated verification.

Experimental Conditions. We compared five conditions: (1) **No-Guidance** (the proxy student works independently), (2) **Basic-Guidance** (a tutor using a generic Socratic prompt [32]), (3) **TreeIn-**

¹Source code: <https://github.com/Snail664/coding-tutor>

Table 2

Comparative Dialogue Analysis: TreeInstruct’s Linear Questioning vs. SHARP’s Stateful Debugging

TreeInstruct	SHARP
Instructor: What type should nodes_sum be if you want to perform arithmetic operations like addition with it?	Instructor: What do you notice about the way nodes_sum is initialized and how it’s then used within the loop for arithmetic operations?
Student: nodes_sum should be an integer type to perform arithmetic operations like addition with it.	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.
Instructor: 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?	Instructor: 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?

struct (the knowledge-space estimation approach of [11]), (4) **SHARP (w.o. Milestone est.)** (our model without milestone progress estimation), and (5) **SHARP (w. Milestone est.)** (the complete model).

Metrics. We employed four metrics: *Success Rate* (proportion of final solutions passing all test cases), *Turns per Bug* (average dialogue turns per bug), *Indirectness* (whether hints avoid revealing direct code solutions, scored 0 or 1 by an LLM judge), and *Relevance* (focus and appropriateness of guidance, scored 1–5).

Results. Table 1 presents the full results. SHARP (w. Milestone est.) achieved the highest success rates across all scenarios (0.98, 0.94, and 0.92 for 1-, 2-, and 3-bug settings) while maintaining the highest Relevance scores (4.94, 4.88, 4.94). Figure 3 visualizes the 3-bug success rates where the performance gap is most evident. The ablation results demonstrate the value of milestone estimation: SHARP (w.o. Milestone est.) already outperformed baselines, and the further improvement of the full model (e.g., 0.92 vs. 0.88 in the 3-bug case) shows the direct impact of accurate progress estimation on hint relevance.

Dialogue Quality Analysis. As shown in Table 2, TreeInstruct’s stateless design leads to an unnatural conversation flow where correct student answers are not acknowledged, potentially causing missed bug fixes. In contrast, SHARP’s stateful, acknowledgment-driven approach maintains a coherent, productive problem-solving flow.

5.2. Exploratory Classroom Pilot Study

Beyond controlled benchmarks, SHARP was piloted with 42 secondary school students at an international school in Hong Kong (RQ3). Initial observations showed a 78% task-completion rate for multi-bug problems, with 85% of students self-reporting improved debugging confidence. Hint utilization correlated with success rates (Pearson’s $r = 0.72$), suggesting effective scaffolding and alignment with benchmark findings.

6. Discussion and Limitations

Our findings demonstrate that a stateful, milestone-driven Socratic tutor can significantly improve success rates in complex debugging tasks compared to stateless or non-personalized approaches. The results reveal a clear trade-off: methods like TreeInstruct minimize interaction turns but sacrifice success rates, while SHARP achieves the highest task completion with only moderate increase in interaction length.

We acknowledge several limitations. First, the LLM proxy evaluation does not capture the complex affective dimensions of human learning. Second, our classroom pilot was exploratory, without a control condition, and did not measure long-term learning gains. Third, while we use the term “Socratic,” our system is more accurately described as providing “Socratic-inspired milestone scaffolding.” Finally, the

student model sometimes fails to incorporate bug fixes from dialogue into final solutions, particularly for problems with multiple interacting bugs.

7. Conclusion and Future Work

We introduced SHARP, a tutoring system that leverages generative AI and real-time execution feedback to provide adaptive, milestone-driven scaffolding for competitive programming. Through a mixed-methods evaluation, we demonstrated that SHARP significantly outperforms baseline models by providing stateful, context-aware guidance. The ablation study confirmed the critical contribution of milestone estimation, and the pilot study provided preliminary evidence of real-world applicability.

Future work will pursue: (1) incremental, bug-by-bug code revision strategies to reduce cognitive load and error propagation; (2) enhanced context management via conversation summarization for improved hint relevance in extended sessions; (3) adaptive milestone granularity tailored to task complexity; and (4) human-in-the-loop integration for ambiguous scenarios. Building on recent work in optimal knowledge tracing for personalized programming education [33], future iterations could incorporate formal student modeling to predict mastery levels and adapt hint difficulty. Controlled studies within K-12 curriculum modules would allow rigorous measurement of learning gains under real classroom conditions. The interaction data collected through Ask Codey's feedback pipeline provides a foundation for future educational data mining research, including modeling problem-solving trajectories and predicting student disengagement.

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] J. Finnie-Ansley, P. Denny, B. A. Becker, A. Luxton-Reilly, J. Prather, The robots are coming: Exploring the implications of OpenAI Codex on introductory programming, in: Proceedings of the 24th Australasian computing education conference, 2022, pp. 10–19.
- [2] 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).
- [3] 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.
- [4] 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.
- [5] Y. Wang, S. Guo, C. W. Tan, From code generation to software testing: AI copilot with context-based retrieval-augmented generation, *IEEE Software* 42 (2025) 34–42. doi:10.1109/MS.2025.3549628.
- [6] M. F. Wong, C. W. Tan, Aligning crowd-sourced human feedback for reinforcement learning on code generation by large language models, *IEEE Transactions on Big Data* (2025). doi:10.1109/TBDATA.2024.3524104, early access.

- [7] A. El-Kishky, A. Wei, A. Saraiva, B. Minaiev, D. Selsam, D. Dohan, F. Song, H. Lightman, I. Clavera, J. Pachocki, et al., Competitive programming with large reasoning models, *arXiv preprint arXiv:2502.06807* (2025).
- [8] R. S. Lindberg, T. H. Laine, L. Haaranen, Gamifying programming education in k-12: A review of programming curricula in seven countries and programming games, *British journal of educational technology* 50 (2019) 1979–1995.
- [9] J. Sweller, Cognitive load during problem solving: Effects on learning, *Cognitive science* 12 (1988) 257–285.
- [10] B. D. Lund, T. Wang, N. R. Mannuru, B. Nie, S. Shimray, Z. Wang, Chatgpt and a new academic reality: Artificial intelligence-written research papers and the ethics of the large language models in scholarly publishing, *Journal of the Association for Information Science and Technology* 74 (2023) 570–581. doi:10.1002/asi.24750.
- [11] 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>.
- [12] J. Paladines, J. Ramirez, A systematic literature review of intelligent tutoring systems with dialogue in natural language, *IEEE Access* 8 (2020) 164246–164267.
- [13] A. C. Graesser, M. W. Conley, A. Olney, Intelligent tutoring systems. (2012).
- [14] H. Keuning, J. Jeuring, J. Barendsen, A systematic literature review of automated feedback generation for programming exercises, *ACM Transactions on Computing Education (TOCE)* 19 (2018) 1–43.
- [15] T. Crow, A. Luxton-Reilly, B. Wuensche, Intelligent tutoring systems for programming education: a systematic review, in: *Proceedings of the 20th Australasian computing education conference*, 2018, pp. 53–62.
- [16] J. McBroom, I. Koprinska, K. Yacef, A survey of automated programming hint generation: The hints framework, *ACM Computing Surveys (CSUR)* 54 (2021) 1–27.
- [17] 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) 100327.
- [18] A. W. Fodouop Kouam, et al., The effectiveness of intelligent tutoring systems in supporting students with varying levels of programming experience, *Discover Education* 3 (2024) 385.
- [19] G. Jošt, V. Taneski, S. Karakatič, The impact of large language models on programming education and student learning outcomes, *Applied Sciences* 14 (2024) 4115.
- [20] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günemann, E. Hüllermeier, et al., ChatGPT for good? on opportunities and challenges of large language models for education, *Learning and individual differences* 103 (2023) 102274.
- [21] L. Elder, R. Paul, The role of Socratic questioning in thinking, teaching, and learning, *The Clearing House* 71 (1998) 297–301.
- [22] G. Kestin, et al., Ai tutoring outperforms in-class active learning, *Scientific Reports* 15 (2025).
- [23] 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).
- [24] L. Roest, H. Keuning, J. Jeuring, Next-step hint generation for introductory programming using large language models, in: *Proceedings of the 26th Australasian Computing Education Conference*, ACM, 2024, pp. 59–68.
- [25] T. Phung, V.-A. Pădurean, A. Singh, C. Brooks, J. Stamper, M. Kazemitabaar, Automating human tutor-style programming feedback: Leveraging gpt-4 tutor model for hint generation and gpt-3.5 student model for hint validation, in: *Proceedings of the 14th Learning Analytics and Knowledge Conference*, ACM, 2024, pp. 54–65.
- [26] R. Xiao, X. Hou, J. Stamper, 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*, 2024, pp. 1–10.

- [27] T. W. Price, Y. Dong, D. Lipovac, isnap: towards intelligent tutoring in novice programming environments, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on computer science education, 2017, pp. 483–488.
- [28] Y. Wang, S. Guo, L. Ling, C. W. Tan, Nemobot: Crafting strategic gaming llm agents for k-12 ai education, in: Proceedings of the Eleventh ACM Conference on Learning@ Scale, 2024, pp. 393–397.
- [29] A. C. Graesser, K. Wiemer-Hastings, P. Wiemer-Hastings, R. Kreuz, T. R. Group, et al., Autotutor: A simulation of a human tutor, *Cognitive Systems Research* 1 (1999) 35–51.
- [30] 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.
- [31] 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.
- [32] E. Al-Hossami, R. Bunescu, R. Teehan, L. Powell, K. Mahajan, M. Dorodchi, Socratic questioning of novice debuggers: A benchmark dataset and preliminary evaluations, in: Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023), 2023, pp. 709–726.
- [33] Y. Wang, P.-D. Yu, C. W. Tan, Future-proofing programmers: Optimal knowledge tracing for ai-assisted personalized education, arXiv preprint arXiv:2509.23996 (2025).