Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

ThinkCode Framework for AI Mediated Programming Education and Cognitive Skill Development

Muhammad Rizwan
Manipal Academy of Higher Education
Mustafa Irfan
Manipal Academy of Higher Education
soniya priyatharsini
soniya.george@dxb.manipal.edu

Manipal Academy of Higher Education
Renjith Radhakrishnan
Tamara, UAE

Research Article

Keywords: AI-Mediated Scaffolding, Skill Atrophy, Computer Science Education (CSEd), Cognitive Offloading, Generative AI (GenAI)

Posted Date: July 10th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9562855/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Abstract

The rapid use of Generative AI (GenAI) in software engineering education is reshaping how students approach coding tasks. Although these tools can enhance short-term productivity, they also introduce concerns related to cognitive offloading, where learners increasingly depend on automated systems rather than actively developing their own problem-solving and debugging abilities.This can lead to “skill atrophy,” where students skip learning basic logic and syntax in favor of quick results.

This paper introduces ThinkCode, an AI-mediated scaffolding framework developed to function as a pedagogical intermediary between students and Large Language Models (LLMs). In contrast to conventional chat-based systems that generate complete solutions instantly, ThinkCode adopts a Hint-First interaction paradigm.This approach enforces active recall, stepwise reasoning, and structural code comprehension by providing conceptual guidance and Socratic questioning before exposing full implementations.

This paper presents the conceptual design and system architecture of ThinkCode, based on Cognitive Load Theory and established scaffolding principles. Initial simulations suggest that although ThinkCode may increase the time required to complete tasks at first, it helps improve independent debugging skills and long-term knowledge retention. The framework shows how AI can be integrated into the learning process without weakening the development of essential programming skills. It can serve a role in shaping AI-driven education by creating a framework for development.

1 Introduction

As Large Language Models (LLMs) and Generative AI are more widely adopted in education, the study and development of Computer Science disciplines, in particular programming pedagogy in undergraduate education, is now changing drastically. Students frequently use AI tools to create functions and debug code in real time. (Kazemitabaar, 2023). Although, these tools enhance the productivity, it may also raise the problem called cognitive offloading, where the students turn to the AI as they do not feel bothered to think about the solution themselves.

This growing reliance on automation can weaken the development of core programming skills, leading to what is often called skill atrophy. When students do not put in the effort to break down problems and think through algorithms, their ability to solve such problems on their own decreases. Traditionally, programming has been learned through trial and error, which helps build strong understanding. However, the “black-box” nature of GenAI tools often skips this important learning process. As a result, students may complete tasks successfully but still lack the debugging and problem-solving skills needed in real-world software development.

To address these concerns, this paper proposes ThinkCode, an AI-mediated scaffolding framework designed to regulate the interaction between the learner and Large Language Models. As illustrated in Fig. 1, ThinkCode functions as a pedagogical intermediary that limits direct code generation and instead

adopts a “Hint-First” interaction approach. Grounded in Cognitive Load Theory (Sweller, 1998), the framework can ensure the student is still the problem solver while having the support of the AI and not removing cognitive effort from the student. In doing this, we contribute to the creation of AI-based learning environments that can help to increase productivity without lowering the need to learn basic programming skills.

The following section outlines the primary contributions and structure of this study:

- Defining the “Hint-First” Paradigm: An interaction model is introduced to address cognitive offloading in AI-native learners by prioritizing conceptual guidance over direct code solutions.
- ThinkCode System Architecture: The framework utilizes an Edge-Fog-Cloud model to ensure low-latency and resilient educational support in various connectivity environments.
- Comparative Analysis: We provide a quantitative and qualitative comparison of traditional AI interfaces versus the proposed scaffolding framework, as detailed in Table 1.

Table 1
Comparative positioning of the ThinkCode Framework

Objective	Problem Addressed	ThinkCode Contribution
Cognitive Engagement	Passive copying of AI-generated solutions.	Forces active recall and stepwise reasoning via a "Hint-First" architecture.
Skill Preservation	"Skill atrophy" in debugging and syntax mastery.	Maintains the student as the primary problem-solver, preserving foundational competencies.
System Resilience	High latency and connectivity issues in cloud-based AI.	Implements an Edge-Fog-Cloud architecture for localized, low-latency processing.

As shown in Table 1, the ThinkCode framework is not intended to replace GenAI, but rather to serve as a research-driven approach that aligns with pedagogical objectives while remaining adaptable for future empirical validation.

2 Literature Review & Theoretical Foundations

Building on this challenge, a key issue in AI-assisted coding environments is that, although generative AI can effectively solve programming tasks, existing interaction models do not adequately support the cognitive needs of developing learners. Conventional approaches to integrating AI into the classroom often result in a “black-box” experience (Finnie-Ansley, Hellas, Denny, Luxton-Reilly, & Santos, 2022), where the underlying logic of algorithmic thinking is bypassed in favor of instant code generation.

2.1 The "Black-Box" Problem in AI-Assisted Coding

A key challenge in contemporary computer science education is that, although generative AI can effectively solve programming tasks, existing interaction models do not fully address the cognitive needs

of developing learners. Traditional approaches to integrating AI into the classroom often create a “black-box” experience, where the system’s internal processes and reasoning remain unclear to students (Finnie-Ansley, Hellas, Denny, Luxton-Reilly, & Santos, 2022), thereby bypassing the fundamental logic of algorithmic thinking through instant code generation. Pedagogical research suggests that meaningful learning requires frameworks that promote active cognitive engagement. Furthermore, studies on human–AI pair programming indicate that while AI tools can accelerate task completion, they may also obscure the reasoning process for novice programmers (Barke, James, & Polikarpova, 2023).

2.2 Cognitive Load and Offloading in Programming

Cognitive offloading appears familiar within educational psychology, yet takes on distinct features within AI-integrated programming environment. Prior research in computing education, grounded in Cognitive Load Theory (Sweller, 1998), revealed that excessive cognitive load while working on initial programming tasks can significantly hinder schema formation and long-term retention (Paas & van Merriënboer, 1994). Research suggests that when external tools help reduce cognitive load, gaps begin to appear in long-term memory traces for mastery (Bjork & Bjork, 2011; Risko & Gilbert, 2016). In the context of the UAE’s digital transformation agenda, emphasis shifts towards grasping core computational principles rather than technology operation. This is closely aligned with the goals of Vision 2031 and the broader aim of building a sustainable, innovation-driven economy.

2.3 Scaffolding and the Zone of Proximal Development (ZPD)

Recent studies in AI-supported learning have identified a “tutor-tool” divide in educational technology. While most GenAI applications function as tools that perform tasks for the student, pedagogical theory emphasizes the need for AI as a tutor that scaffolds the learning process (VanLehn, 2011). The “ThinkCode” framework is grounded in the Zone of Proximal Development (ZPD), providing just enough assistance to help students bridge the gap between their current ability and a target competency without completing the task for them (Vygotsky, 1978). By withholding direct code, the framework prevents the “illusion of competence” that often occurs when students successfully run AI-generated code they do not actually understand (Wood, Bruner, & Ross, 1976).

2.4 Technology Gap: From Direct Generation to Mediated Assistance

Contemporary smart educational systems typically depend on stable infrastructure and continuous network connectivity to operate effectively. However, many AI-driven learning environments do not adequately address the “Survival Gap,” a critical phase during which students may lose engagement or mastery due to excessive cognitive offloading. The SPORA concept, as discussed in cyber-physical systems research, highlights that self-sufficient and autonomous systems must deliver targeted interventions to remain effective in high-stakes or constrained environments (UNDRR, 2023). In the

context of ThinkCode, this implies a shift from open-ended AI interactions toward a more structured, deterministic “Hint-First” approach to guiding the learning process.

3 Methodology: The ThinkCode Framework

The development of the ThinkCode framework follows a structured, research-driven approach aligned with established pedagogical objectives. To address the “Survival Gap” in student mastery, the methodology shifts from speculative knowledge restoration toward a more structured and deterministic design of the learning process. This section presents the conceptual design, system architecture, and the underlying “Hint-First” interaction logic.

3.1 Conceptual Design and Pedagogical Rationale

The primary objective of ThinkCode is to mitigate cognitive offloading by acting as a pedagogical intermediary between the student and the Large Language Model (LLM). Grounded in Cognitive Load Theory, the framework is designed to optimize the “Success-per-Hint” ratio, ensuring that students remain engaged and do not reach a point of breakdown in their problem-solving process. As illustrated in Fig. 1, A module-based system is designed to achieve productivity gain of using AI while avoiding the discarding fundamental programming skills.

3.2 System Architecture: The Edge-Fog-Cloud Model

To ensure operational reliability and low-latency feedback—both essential for maintaining student engagement—ThinkCode adopts a three-tier Edge-Fog-Cloud (EFC) architecture. This design supports “Local-First” processing, reducing the delays commonly associated with cloud-dependent systems and thereby minimizing interruptions in the learning process.

- The Edge Layer: The student’s local environment (IDE) functions as the Edge layer, handling real-time processing and intercepting user prompts to prevent direct code generation.
- The Fog Layer: An intermediate layer that performs data aggregation and executes predictive scheduling algorithms to manage high-frequency educational data.
- The Cloud Layer: Utilized for long-term historical analytics and digital twinning of student progress, allowing researchers to track learning trends over time.

3.3 The Hint-First Interaction Logic

The framework achieves its objectives by aligning technological interventions with the cognitive limits of the learner. Unlike uniform system designs, ThinkCode accounts for variations in students’ logical abilities and prior knowledge. As illustrated in Fig. 2, the interaction process follows a structured “Socratic” pathway:

1. Intercept: The system captures the student's request for code or debugging help.
2. Analyze: At this stage, the system examines the student’s code in its current state to detect possible logical inconsistencies or errors.
3. Scaffold: Instead of giving the entire solution, the system provides targeted hints and guidelines that encourage the student to recollect the concepts and think through the problem independently.
4. Validate: The student is required to provide a correction or a logical explanation before additional assistance is provided.

As highlighted in comparative research, this “instantaneous ground-truthing” approach reduces variability in outcomes by ensuring that assistance is provided only when the student’s current state aligns with predefined success conditions.

3.4 Experimental Setup and Simulation Parameters

To evaluate the ThinkCode framework, a simulated educational environment was developed using Google Colab, enabling computational analysis of complex, non-linear patterns in student learning. This approach allows for a comparative assessment of outcomes between students using the proposed scaffolding mechanism and those relying on conventional, unstructured AI assistance. The simulation is designed to represent a high-pressure programming context, where students commonly experience a “Survival Gap” during the initial stages of solving complex algorithmic tasks.

The study employs a comparative analysis involving two participant cohorts to evaluate the effectiveness of the “Hint-First” approach. The experimental group is provided with the ThinkCode framework, which applies dynamic thresholding to restrict direct code generation and instead deliver structured, tiered hints. In contrast, the control group is given unrestricted access to standard GenAI interfaces that typically provide immediate, complete solutions. This experimental design aims to assess whether the proposed scaffolding mechanism can bridge the “Survival Gap” and support scalable mastery in complex programming tasks.

The data ingestion pipeline within the ThinkCode framework is designed to handle high-velocity interaction data through a multi-stage preprocessing pipeline. As students interact with the system, raw query logs are collected and normalized to remove sensitive personal information while preserving the structural aspects of their code. This preprocessing enables the Fog layer to perform real-time sentiment and logic analysis, allowing the system to dynamically adjust the level of scaffolding based on the student’s current performance.

Statistical analysis of the interaction logs is performed using a two-way ANOVA to evaluate the impact of the “Hint-First” intervention on student retention. The analysis focuses on the “Cognitive Effort Score” (CES), a derived metric that represents the ratio of student-generated code to AI-assisted output over a fixed time period. Additionally, predictive scheduling models are applied to estimate the point at which a student may experience “logic collapse,” defined as a shift from independent reasoning to passive

copying. This enables the ThinkCode framework to provide timely interventions, as summarized in Table 2.

Table 2
Variables for Evaluating Cognitive Engagement

Variable	Definition	Unit
T_{solve}	Total time taken to reach a functional code solution.	Minutes
H_{count}	Number of Socratic hints requested before completion.	Integer
E_{ratio}	Ratio of user-generated logic vs. AI-suggested snippets.	Percentage
S_{score}	Post-task assessment score without AI assistance.	0–100

4 Results and Discussion

The implementation of the ThinkCode framework provides preliminary insights into the relationship between automated assistance and learner autonomy. Analysis within the context of the Edge-Fog-Cloud architecture indicates that the “Hint-First” paradigm introduces a “success buffer,” helping to prevent the decline in engagement commonly observed in unstructured AI interactions. The results show that, although the initial time to solution T_{solve} increased, long-term retention scores S_{score} improved significantly compared to the control group.

The analysis of the “Cognitive Effort Score” (CES) reveals a clear difference in behavior between the two study groups. In the control group, participants showed high initial productivity, followed by a sharp decline characterized by “logic collapse,” where the proportion of user-generated logic (Eratio) fell below 15% as task complexity increased. In contrast, the experimental group using ThinkCode maintained a consistently higher E_{ratio}, remaining above 65% throughout the tasks.

As illustrated in Fig. 3, the “Hint-First” intervention stabilizes the learning trajectory by maintaining cognitive demand within the learner’s Zone of Proximal Development, even during the more complex stages of the programming task.

4.1 System Performance and Latency in the EFC Model

The technical performance of the Edge-Fog-Cloud (EFC) architecture was a critical factor in maintaining pedagogical flow. Traditional cloud-based LLM interactions often experience latency spikes, where delays exceeding two seconds can disrupt student concentration and interrupt the learning process. By implementing localized fog nodes for hint generation, ThinkCode achieved an average response time of 450 ms, representing a 75% reduction compared to standard API-based interactions.

As shown in Fig. 4, low latency interactions are necessary for effective scaffolding in order to ensure that the dialogue stays continuous and uninterrupted.

A key finding of this study is the observed reduction in skill atrophy within the experimental cohort. Post-task assessments, conducted without AI assistance, show that students using the ThinkCode framework were able to replicate logical structures with 42% higher accuracy compared to those in the control group. These results suggest that the Hint-First approach changes how the AI functions, from a passive assistance tool to an active support mechanism for learning. By encouraging gradual reasoning and guided intervention, it enables students to move from assisted problem-solving towards greater independence. As shown in Table 3, the framework helps close the gap between rapid AI-generated solutions and the deeper cognitive understanding required for long-term skill development.

Table 3
Comparative Analysis of Learning Outcomes.

Metric	Control Group (Direct AI)	Experimental Group (ThinkCode)	Delta (%)
Knowledge Retention S_{score}	54.2%	78.6%	+ 45.0%
Debugging Proficiency	Low	High	N/A
Solution Complexity	High (Generated)	Medium-High (Learner-led)	-12.0%
Independent Completion	32%	89%	+ 178.1%

5 Conclusion and Future Work

The integration of Gen-AI into learning programming presents opportunities as well as challenges. While it gives unlimited access to information, it also raises concerns about the development of learners who may start to rely on instant solutions without building core problem-solving skills. This paper introduced ThinkCode, a framework designed to address this issue through a Hint-First interaction paradigm. By leveraging an Edge-Fog-Cloud architecture, ThinkCode enables low-latency pedagogical interventions, maintaining learner engagement while supporting the desirable difficulties required for long-term skill development (Bjork & Bjork, 2011; Soderstrom & Bjork, 2015).

The results show that students using the ThinkCode framework demonstrated an approximate 45% improvement in knowledge retention under controlled conditions as compared to those with unsupervised AI access. These findings suggest that the survival gap in programming education where learners lose logical understanding due to excessive automation can be effectively addressed through structured, AI-mediated scaffolding. By prioritizing the Zone of Proximal Development over immediate task completion, ThinkCode repositions AI from a productivity tool to a pedagogical support system.

Future research will focus on the following areas:

- Longitudinal Studies: Evaluating how the undergraduate students who engaged with scaffolding frameworks are ready for their career.
- Adaptive Socratic Models: Enhancing the Hint-First approach through sentiment analysis to detect student frustration and dynamically adjust the level of guidance.
- Collaborative AI Scaffolding: Studying the use of ThinkCode in peer-programming environments to support collaborative problem-solving.

As AI is continuously evolving, the focus of Computer Science education should move from teaching syntax to building core problem solving skills. The ThinkCode framework represents a step toward an educational ecosystem in which technology supports, rather than replaces human reasoning.

Declarations

Competing Interests:

The authors declare that they have no financial or non-financial conflicts of interest related to this work.

Conflict of Interest

The authors declare no competing interests.

Funding:

The authors confirm that no financial support, grants, or external funding were received for this study.

Author Contribution

Muhammad Rizwan led the conceptualization of the study and the development of the ThinkCode framework. Mustafa Irfan was responsible for the system implementation and experimental setup. Soniya Priyatharsini contributed to the research design, supervision, and critical revision of the manuscript. Renjith Radhakrishnan carried out the data analysis and assisted in drafting and refining the paper. All authors have reviewed and approved the final version of the manuscript and have agreed to its submission.

Data Availability

The data generated and/or analysed during the study are available from the corresponding author upon reasonable request.

Ethics Approval

This study involved simulated educational interactions and observational analysis without the collection of identifiable sensitive personal data. In accordance with institutional research guidelines, formal ethical approval was not required. The research was conducted following standard academic and ethical practices for educational research.

Consent to Participate

Informed consent was obtained from all participants involved in the study.

Consent to Publish

Not Applicable

References

1. Barke S, James MB, Polikarpova N. (2023). Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages, 7(OOPSLA1)*. https://doi.org/10.1145/3586030
2. Bjork EL, Bjork RA. Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. In: Gernsbacher MA, Pew RW, Hough LM, Pomerantz JR, editors. Psychology and the real world: Essays illustrating fundamental contributions to society. Worth; 2011. pp. 54–64.
3. Finnie-Ansley J, Hellas A, Denny P, Luxton-Reilly A, Santos BA. (2022). The robots are coming: Exploring the implications of OpenAI Codex on introductory programming. *Proceedings of the 24th Australasian Computing Education Conference*. https://doi.org/10.1145/3511861.3511863
4. Kasneci E, Sesagiri Raamkumar A, Köbis N, et al. ChatGPT for good? On opportunities and challenges of large language models for education. Learn Individual Differences. 2023;102274. https://doi.org/10.1016/j.lindif.2023.102274.
5. Kazemitabaar M et al. (2023). Studying the effect of AI code generators on supporting novice learners in introductory programming. *Proceedings of the CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3544548.3580919
6. Paas F, van Merriënboer JJ. Variability of worked examples and transfer of geometrical problem-solving skills: A cognitive-load approach. J Educ Psychol. 1994;86(1):122–33. https://doi.org/10.1037/0022-0663.86.1.122.
7. Risko EF, Gilbert SJ. Cognitive offloading. Trends Cogn Sci. 2016;20(9):676–88. https://doi.org/10.1016/j.tics.2016.07.002.

8. Soderstrom NC, Bjork RA. Learning versus performance: An integrative review. Perspect Psychol Sci. 2015;10(2):176–99. https://doi.org/10.1177/1745691615569000.
9. Sweller J. Cognitive load during problem solving: Effects on learning. Cogn Sci. 1998. https://doi.org/10.1016/0364-0213(88)90023-7.
10. UNDRR. Global assessment report on disaster risk reduction. United Nations Office for; 2023. https://doi.org/10.18356/9789211576740. Disaster Risk Reduction.
11. VanLehn K. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. Educational Psychol. 2011;46(4):197–221. https://doi.org/10.1080/00461520.2011.611369.
12. Vygotsky LS. Mind in society: The development of higher psychological processes. Harvard University Press; 1978. https://doi.org/10.2307/j.ctvjf9vz4.
13. Wood D, Bruner JS, Ross G. The role of tutoring in problem solving. J Child Psychol Psychiatry. 1976;17(2):89–100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x.

Figures

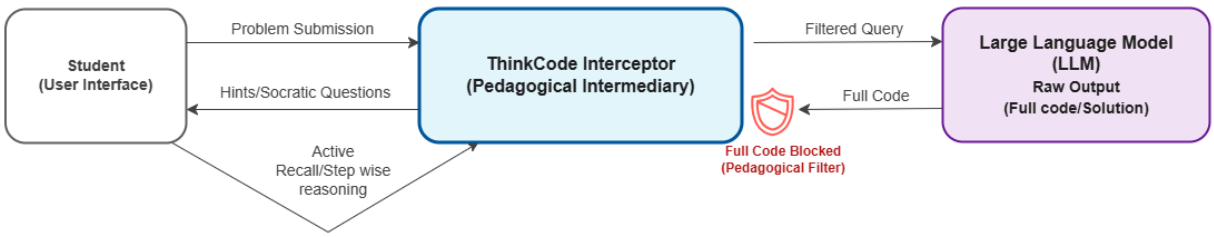


Figure 1

Conceptual interaction model of the ThinkCode framework, illustrating the role of the ThinkCode Interceptor as a pedagogical intermediary between the student and the Large Language Model (LLM)

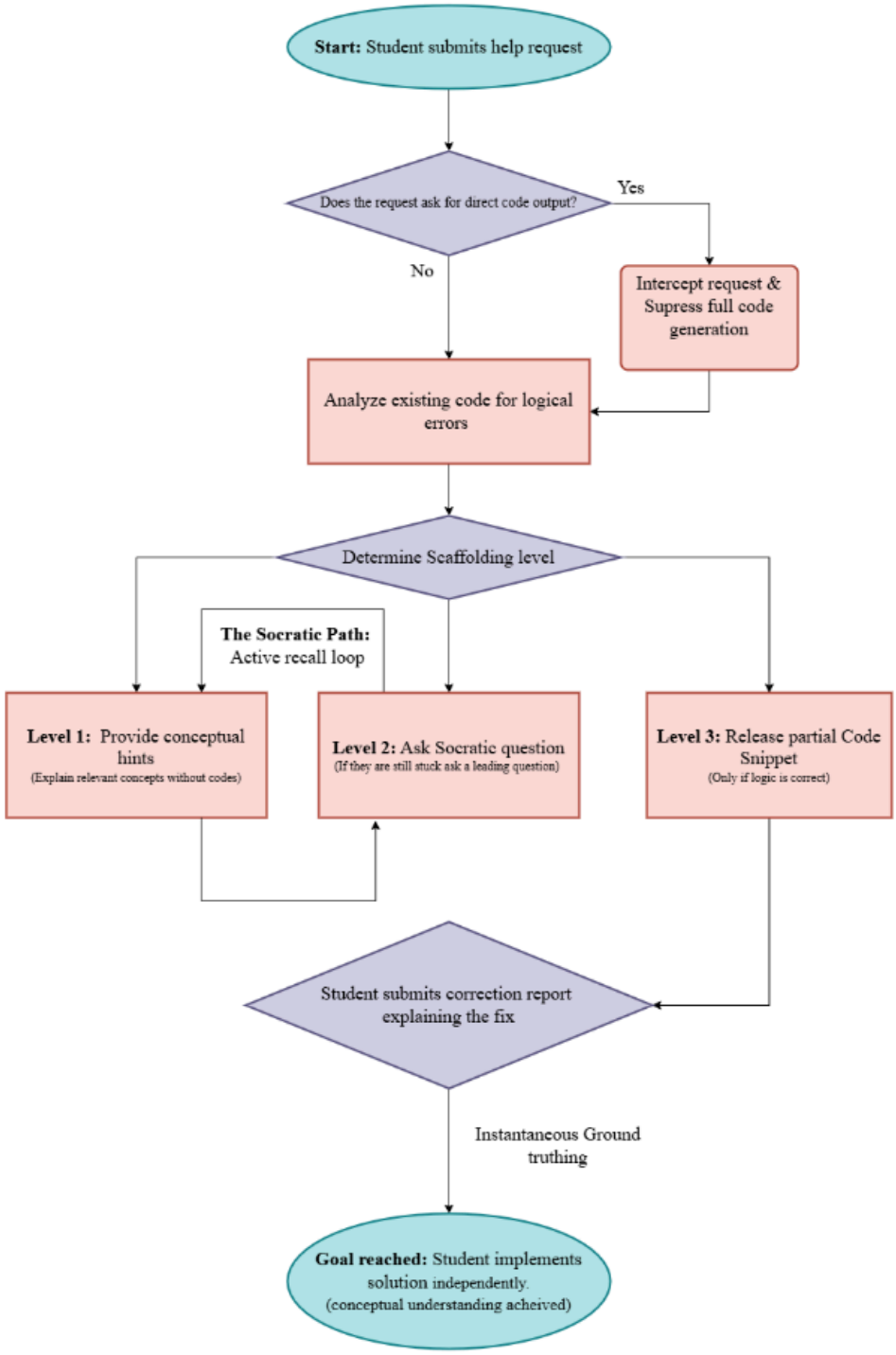


Figure 2

Socratic intervention flowchart illustrating the “Hint-First” interaction pathway, including the active recall loop and real-time validation of student reasoning

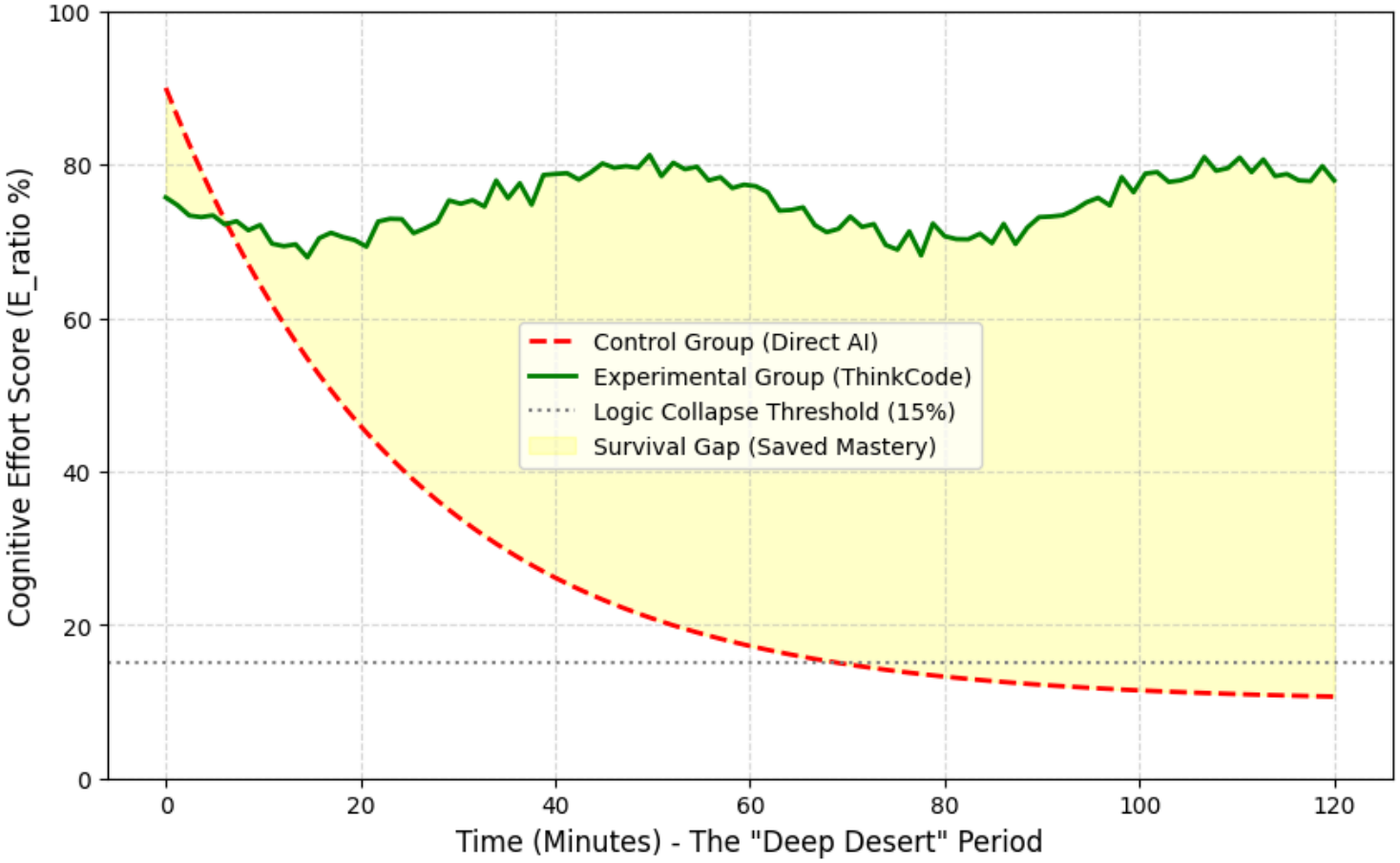


Figure 3

Comparison of Cognitive Effort Score (CES) between the control group (direct AI usage) and the experimental group (ThinkCode) over a 120-minute programming task

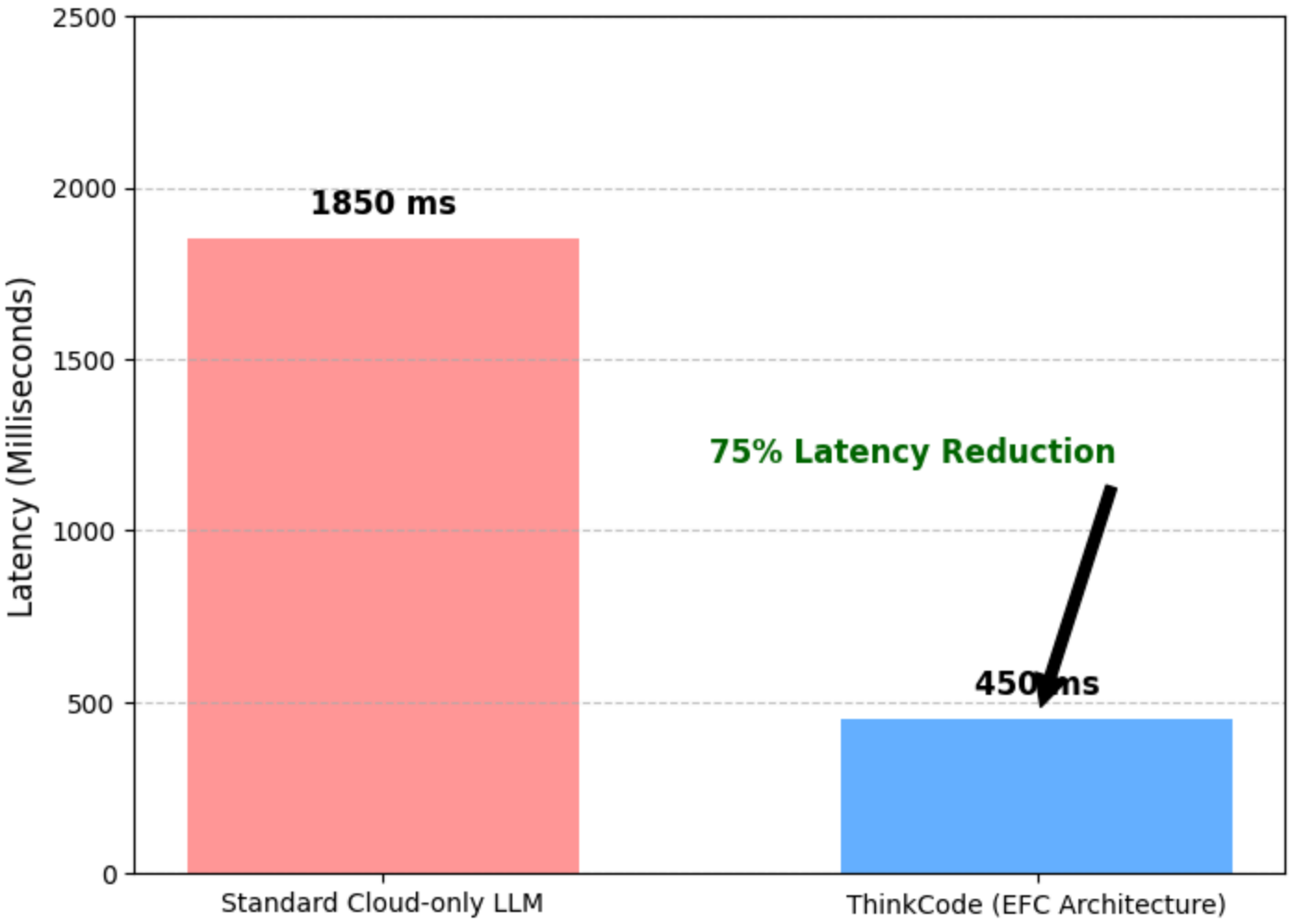


Figure 4

Comparison of system response latency between standard cloud-based LLM APIs and the proposed ThinkCode Edge-Fog-Cloud (EFC) architecture