Design and Pilot Evaluation of SmartCode Tutor (SCT): A Mixed-Methods Framework for AI-Assisted Introductory Programming in Higher Education

Nor Zakiah Lamin[1], Wan Nor Asnida Wan Jusoh[2], Wan Asiah Wan Muhamad Tahir[3], Siti Faizah Miserom[4], Abdullah Mohd Zin[5]

[1,2,3,4] Universiti Poly-Tech Malaysia, Faculty of Computing and Multimedia
Jalan 6/91, Taman Shamelin Perkasa, 56100 Cheras, Kuala Lumpur, Malaysia.
nzakiah@uptm.edu.my*, wnasnida@uptm.edu.my,
wanasiah@uptm.edu.my faizah@uptm.edu.my

[5] Al-Madinah International University, Faculty of Computer and Information Technology
Pusat Perdagangan Salak II, No. 18, Jalan 2/125e, Taman Desa Petaling, 57100 Kuala Lumpur. Malaysia
abdullah@uptm.edu.my

Abstract. Generative AI and AI coding assistants such as large language model–based tutors and IDE copilots can provide immediate, personalized feedback for novice programmers, but unstructured use risks shallow learning and academic integrity issues. This paper presents SmartCode Tutor (SCT), a structured, AI-enhanced framework that combines scaffolded prompt templates, automated formative feedback, and lightweight learning analytics to support introductory programming. We report a pilot, mixed-methods, single-group pretest–posttest study conducted in an undergraduate programming course in Malaysia. Quantitative evidence from pre/post assessments and interaction logs was complemented with student and instructor questionnaires and short reflective interviews. The pilot indicates that SCT approach is feasible to integrate into weekly labs and helps students progress from syntax-level troubleshooting to more systematic debugging and explanation-based problem solving, while providing instructors with actionable indicators of common misconceptions. We also describe the academic-integrity safeguards embedded in SCT (answer-avoidance prompting, citation and attribution guidance, and policy-aligned usage rules). The contributions of this study are an SCT conceptual model grounded in scaffolding and cognitive load principles; an operational workflow that can be adopted with common tools (LLM chatbot + repository/IDE); and pilot evidence and lessons learned to inform a larger controlled study.

Keywords: Generative AI; AI coding assistants; programming education; automated feedback; learning analytics.

N. A. Ishak et al. (eds.), *Proceedings of the International Conference on Cross- Disciplinary Academic Research 2025 - Track 1 Advances in Computing, Electronics, Engineering, and Mathematics (ICAR-T1 2025)*, Advances in Engineering Research 296,
https://doi.org/10.2991/978-94-6239-636-4_8

1 Introduction

Introductory programming is a foundational course for computing students and an increasingly common requirement for non-computing disciplines. Yet novices often struggle with problem decomposition, syntax, and systematic debugging, which contributes to high cognitive load, frustration, and attrition. Instructors face a scalability problem: frequent practice is essential, but providing timely, individualized feedback on code and misconceptions is labor-intensive in large classes.

At the same time, AI coding assistants and large language models (LLMs) have rapidly become accessible to students. Empirical work shows that such tools can speed up task completion, but they may also generate plausible-looking yet incorrect solutions that novices are not able to evaluate, raising risks of brittle understanding and over-reliance (Dakhel et al., 2023; Imai, 2022). Computing-education scholarship therefore calls for pedagogically guided adoption—tools should support explanation, hinting, and reflection rather than simply producing final answers (Denny et al., 2023; Messer et al., 2024).

However, current classroom implementations are often ad-hoc where students use chatbots or copilots without structured prompts, without alignment to learning outcomes, and without clear integrity guidance. The literature also reports uneven evidence on learning gains—performance improvements do not always translate into deeper conceptual understanding (Bassner et al., 2025). There is a need for an implementable framework that operationalizes scaffolding principles for AI-mediated feedback, integrates lightweight learning analytics for instructors, and embeds academic-integrity safeguards (Messer et al., 2024; Kofinas et al., 2025).

This paper addresses that need by presenting SmartCode Tutor (SCT) approach, a structured AI-enhanced framework for introductory programming. We report a pilot deployment in an undergraduate course in Malaysia and answer the following research questions (RQs):

RQ1. How does SCT influence students' programming performance and debugging efficiency?

RQ2. How do students perceive the usefulness, ease of use, and learning support of SCT?

RQ3. How do instructors perceive SCT in terms of feasibility and workload support?

The main contributions are:

(1) a conceptual SCT model grounded in scaffolding and cognitive load theory.
(2) an operational workflow combining LLM-based feedback, code-quality checks, and learning-analytics indicators; and
(3) pilot evidence and design lessons to inform a larger, controlled evaluation study.

2 Related Works

This section synthesizes recent evidence on AI coding assistants and LLM-based tutoring for novices, automated assessment and feedback in programming education,

and academic-integrity challenges. The synthesis is used to position SCT as a pedagogically structured and integrity-aware approach.

2.1 AI coding assistants and LLM-based tutoring

Studies on IDE copilots and code-generation models show strong productivity potential but warn that novices may accept buggy or non-idiomatic code and may skip critical reasoning steps (Dakhel et al., 2023; Imai, 2022). Within computing education, researchers argue that the instructional value depends on how students interact with these tools—for example, prompt engineering and conversational debugging can be framed as learning activities when prompts elicit explanations and alternatives rather than direct solutions (Denny et al., 2023). Broader work also discusses how generative AI is reshaping assessment and emphasizes the importance of teaching students to validate, test, and justify AI-suggested solutions (Yusuf et al., 2024; Messer et al., 2024).

2.2 Automated assessment and formative feedback

Automated assessment tools have long supported scalability in programming courses, typically through unit tests, static analysis, and style checks. A recent systematic review highlights that while immediate feedback and multiple resubmissions can improve student satisfaction, many tools provide limited diagnostic feedback and rarely address code readability and maintainability (Messer et al., 2024). LLMs open new opportunities to generate richer feedback and rubric-aligned explanations, including semi-automated grading and feedback generation (Yousef et al., 2025), but this requires careful constraints to avoid revealing complete answers.

2.3 Learning analytics for misconceptions and support

Learning analytics can complement automated feedback by identifying error patterns, misconceptions, and at-risk learners from submission logs and interaction traces. For programming, actionable analytics often focus on frequent compiler/runtime errors, inefficient iteration strategies, and prolonged debugging cycles that signal conceptual gaps. Integrating such indicators into instructional decision-making remains an open challenge, particularly when new AI tools change how students produce and revise code (Messer et al., 2024).

2.4 Academic integrity and responsible GenAI use

GenAI creates an integrity dilemma where it can support learning through explanations and examples, but it can also be used to bypass learning outcomes through answer generation. Evidence suggests that markers may struggle to reliably detect GenAI contributions in authentic assessments (Kofinas et al., 2025). Recent systematic reviews and empirical work therefore recommend explicit policies, transparency and attribution practices, and assessment designs that emphasize process evidence (e.g., reasoning, oral

explanation, or live coding) rather than only final artefacts (Bittle & El-Gayar, 2025; Yusuf et al., 2024).

2.5 Gap and SCT positioning

Across these strands, a consistent gap is the lack of an implementable, classroom-ready framework that operationalizes scaffolding for AI-mediated feedback, connects student interactions to instructor-facing analytics, and embeds integrity safeguards. SCT addresses this gap by combining structured prompts (answer-avoidance, self-explanation, and hint-request patterns), automated checks (tests and static analysis), and learning-analytics indicators to support both student learning and instructor workload.

3 Methodologies

This study adopted a mixed-methods approach involving both quantitative and qualitative data collection to ensure a comprehensive understanding of AI integration in programming education. The research was conducted with 60 undergraduate students enrolled in an introductory programming course. Data collection was carried out through multiple channels: quantitative data was obtained from pre- and post-course coding assessments, automated usage logs of the AI tools, and analysis of student code submissions; qualitative insights were gathered from structured surveys, semi-structured interviews with lecturers, and classroom observations. The overall research design is summarized in Fig. 1.

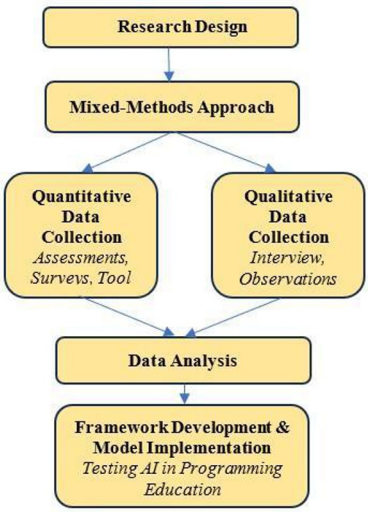


Fig. 1. Research design of methodology

3.1 Research design and context

We conducted a pilot mixed-methods study using a single-group pretest–posttest design. The study took place in an undergraduate introductory programming course (weekly lecture + lab) at a Malaysian higher-education institution. SCT was integrated into laboratory activities as a structured support for writing, testing, and debugging code; students completed pre- and post-assessments aligned to course learning outcomes.

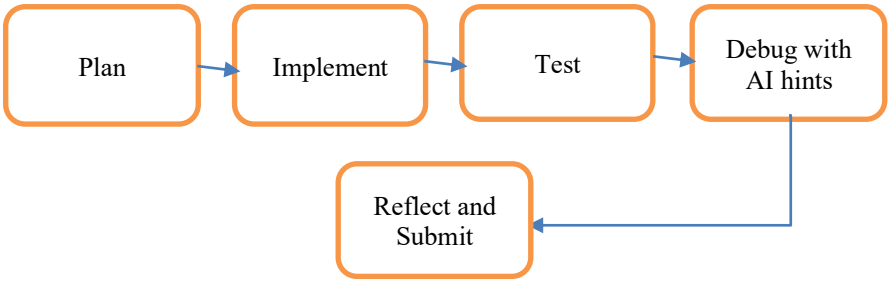
3.2 Participants and sampling

Participants were students enrolled in the course during the study semester. Participation in research data collection was voluntary; course learning activities were not affected by participation decisions. We recorded demographic descriptors at a coarse level (e.g., prior programming exposure) to contextualize findings while minimizing identifiable information.

3.3 SmartCode Tutor (SCT) intervention

SCT operationalizes a repeatable learning loop: where students must understand the problem and articulate assumptions. Next, they need to draft a solution plan/pseudocode and implement code. Then, they need to run tests and inspect errors from the execution. They may request AI hints using answer-avoidance prompt templates, and reflect and document the final rationale. Students interacted with an LLM chatbot for explanations and debugging hints, while code correctness and style were checked using tests and basic static analysis within the development environment. The SCT learning workflow is illustrated in Fig. 2.

Fig. 2. SCT learning workflow



3.4 Instruments and outcome measures

We collected evidence from multiple sources to triangulate learning outcomes and user experience. Firstly, a pre-assessment and post-assessment measuring code comprehension and code writing aligned to course outcomes. Secondly,

programming-activity traces such as compilation errors, test outcomes, and iteration counts extracted from the development workflow. Thirdly, a student questionnaire capturing perceived usefulness, ease of use, learning support, and integrity awareness. Finally, an instructor reflection and short interviews on feasibility and workload. Table 1 summarizes the data sources and analysis mapping.

Table 1. Data sources and analysis mapping

Measure / Data source	Indicator(s)	Analysis approach
Pre/post assessments	Programming performance (comprehension + writing)	Paired comparison; effect size; error profile
Workflow traces	Debugging iterations; common error types; time-on-task proxies	Descriptives + pattern mining; triangulation
Student questionnaire	Usefulness, ease, learning support, integrity awareness	Descriptives; reliability check; correlations (exploratory)
Instructor interview/reflection	Feasibility; workload; observed misconceptions	Thematic analysis

3.5 Procedure

The pilot ran through regular class sessions. Students completed a baseline pre-assessment, then used SCT during weekly lab tasks. Prompt templates were introduced and modelled by the instructor, emphasizing requests for hints, explanations, and test-driven debugging instead of direct full solutions. At the end of the pilot period, students completed the post-assessment and questionnaire; instructors provided reflections and were interviewed.

3.6 Data Analysis

Quantitative data were analyzed using paired comparisons between pre- and post-assessments (parametric or non-parametric depending on normality) and complemented with descriptive summaries of workflow traces. Where applicable, effect sizes were computed to quantify practical significance. Qualitative data (open-ended responses and instructor interviews) were analyzed using reflexive thematic analysis with iterative coding and theme refinement.

3.7 Ethics, tool governance, and integrity safeguards

Ethical approval and informed consent procedures were followed according to institutional requirements. SCT includes integrity safeguards: students are instructed to document AI use, to request explanations or hints rather than final answers, and to

validate outputs through tests. Instructor-provided rubrics and assessment tasks emphasize reasoning and process evidence to reduce answer-only submissions.

4 Results and Discussion

4.1 Implementation feasibility

SCT approach was integrated into existing laboratory sessions without changing core course content. Students were able to follow the structured prompt templates after short instructor modelling, suggesting that the workflow is practical for weekly practice. Instructors reported that the SCT loop helped standardize how students asked for help from problem context to attempted code, error message and request for hint, which reduced back-and-forth clarification during labs.

4.2 Learning outcomes and workflow indicators

Across the pilot, pre- to post-assessment performance improved, and workflow traces indicated fewer repeated trial-and-error cycles as students progressed through labs. Error-trace summaries suggested a shift from syntax-level issues such as missing delimiters toward higher-level logic and boundary-case reasoning, which aligns with the intended scaffolding sequence of SCT. The SCT conceptual model and its mechanisms are shown in Fig. 3.

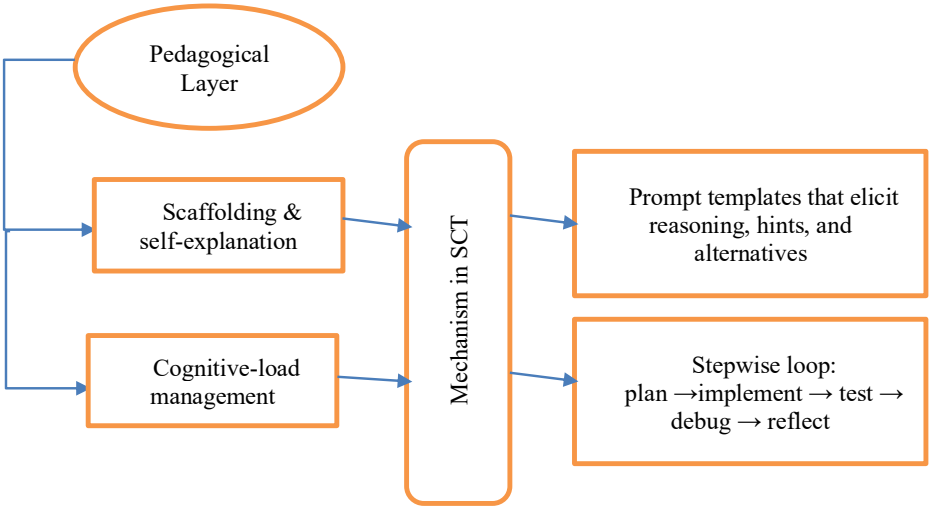


Fig. 3. SCT conceptual model (layers and mechanisms)

4.3 Student Perceptions

Student feedback indicated that SCT approach was most helpful for debugging and for understanding why a solution worked or failed. Students reported that answer-avoidance prompts that "give a hint, not the final code", encouraged them to keep ownership of their solutions. They also noted the importance of validating AI suggestions with tests and compiler feedback, particularly when outputs looked plausible but failed edge cases.

4.4 Instructor perspectives

Instructors highlighted two practical benefits. Firstly, quicker identification of common misconceptions from aggregated error patterns such as loop bounds and off-by-one errors. Second benefits are to reduced repetitive explanations when students used the structured help-seeking format. Concerns included potential over-reliance by weaker students and the need for clear assessment rules and documentation of AI use.

4.5 Discussion

The pilot results support the feasibility of a structured approach to classroom GenAI use. Rather than positioning AI as a solution generator, SCT approach treats AI as a feedback and explanation resource embedded in a test-driven workflow, consistent with calls in computing-education research to focus on process and reasoning (Denny et al., 2023; Yusuf et al., 2024). Our design also addresses limitations noted in automated feedback research—Immediate feedback is valuable, but diagnostic guidance must be aligned to learning outcomes and accompanied by mechanisms that keep students accountable for validation (Messer et al., 2024). Finally, integrity safeguards remain essential given evidence that GenAI contributions can be difficult to detect in authentic assessments (Kofinas et al., 2025).

5 FUTURE WORK

Future work will evaluate SCT approach using a controlled or quasi-experimental design in multi-section comparison with larger cohorts to strengthen causal claims. Beyond paired comparisons, we plan to model adoption and learning mechanisms using advanced techniques such as structural equation modelling (SEM) for technology-acceptance constructs and multilevel models to account for class or section effects. We will also report full psychometric properties of the questionnaire instruments and include longer-term retention measures.

On the learning-analytics side, we will refine indicators that distinguish productive debugging from unproductive trial-and-error and explore early-warning models that combine submission traces with SCT approach interaction patterns. Finally, we will expand integrity safeguards by integrating process-based assessment evidence such as commit history, oral defense, and reflective prompts, and by adapting SCT prompt templates for multilingual learners.

6 CONCLUSIONS

This paper presented SmartCode Tutor (SCT) approach, a structured AI-enhanced framework for introductory programming that combines scaffolded prompt templates, test-driven validation, and instructor-facing learning-analytics indicators. By positioning GenAI as a feedback and explanation tool rather than a solution generator, SCT approach aims to improve help-seeking quality, support systematic debugging, and reduce repetitive instructor workload while maintaining academic integrity.

A pilot mixed-methods deployment suggests that SCT is feasible to integrate into routine labs and can support students' progression toward more reflective debugging practices. These findings are preliminary and motivate a larger controlled evaluation to quantify learning gains and to examine adoption factors across diverse student backgrounds. Overall, SCT contributes a practical, integrity-aware pathway for responsibly incorporating GenAI into programming education.

Acknowledgments. The authors would like to express their sincere gratitude to Universiti Poly-Tech Malaysia (UPTM) for the financial support provided through the Universiti Research Grant (URG). This internal funding has been instrumental in enabling the design, development, and evaluation of the SmartCode Tutor (SCT) Framework and the CodeLift.AI Implementation Model. The support from UPTM has not only facilitated the completion of this research but also contributed to advancing innovative approaches in AI-assisted programming education. The authors also wish to thank the academic colleagues, technical staff, and participating students whose insights and feedback have greatly enriched the outcomes of this work. This study was conducted under the ethical guidelines of Universiti Poly-Tech Malaysia, with informed consent obtained from all participants

Disclosure of Interests. The authors declare that the authors have no competing interest regarding the publication of this manuscript.

References

Bittle, K., & El-Gayar, O.: Generative AI and academic integrity in higher education: A systematic literature review. Information, 16(4), 296. (2025)

Dakhel, A. M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. : GitHub Copilot AI pair programmer: Asset or liability? Journal of Systems and Software, 207, 111734. (2023)

Denny, P., Kumar, V., & Giacaman, N.: Conversing with Copilot: Exploring prompt engineering for solving CS1 problems using natural language. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education (SIGCSE '23), 1136–1142. (2023).

Imai, H. Is GitHub Copilot a substitute for human pair-programming? An empirical study. In Proceedings of the 44th International Conference on Software Engineering (ICSE '22). (2022).

Kofinas, A. K., Pike, D., & Tsay, C. H.H: The impact of generative AI on academic integrity of authentic assessments within a higher education context. British Journal of Educational Technology, 56(6), 2522–2549. (2025)

Messer, M., Brown, N., & Kölling, M.: Automated grading and feedback tools for programming education: A systematic review. ACM Transactions on Computing Education, 24(1). (2024).

Yousef, M., Mohamed, K., Medhat, W., Mohamed, E. H., Khoriba, G., & Arafa, T. BeGrading: Semi-automated grading and feedback generation for programming assignments using a large

language model. International Journal of Artificial Intelligence in Education, 35, 231–255. (2025).
Yusuf, A., Pervin, N., & Román-González, M. Generative AI and the future of higher education: A threat to academic integrity or reformation? Evidence from a multicultural perspective. International Journal of Educational Technology in Higher Education, 21, Article 21. (2024)
Lepp, M., Kaimre,: J Does generative AI help in learning programming: Students' perceptions, reported use and relation to performance. *Computers and Education: Artificial Intelligence* 6, 100642 (2025)
Wang, D., Xie, Z., Lian, D., Lu, Q., Li, N., et al.: The integration of technology and education: An innovative research on the evaluation system of college students' programming ability. *Thinking Skills and Creativity* 2026, 101989 (2026)
Liu, J., Li, S.: Toward artificial intelligence–human paired programming: A review of the educational applications and research on artificial intelligence code-generation tools. *Journal of Educational Computing Research* (2024)
Llerena-Izquierdo, J., Mendez-Reyes, J., Ayala-Carabajo, R., Andrade-Martinez, C.: Innovations in Introductory Programming Education: The Role of AI with Google Colab and Gemini. Education Sciences 14(12), 1330 (2024)