
International Journal of Creative Multimedia

HAI-MinePy: A Human–AI Co-Creation Model for Developing Computational Thinking and Python Coding through Minecraft Education

Goh Kok Ming
kokming888@gmail.com
SJKC Hua Lian 1, Perak, Malaysia
ORCID iD: 0009-0002-6666-2042
(Corresponding Author)

Abstract

Generative artificial intelligence is reshaping coding education by enabling learners to obtain code explanations, translations, algorithms, and debugging support through conversational interaction. However, the pedagogical challenge is to ensure that AI support strengthens, rather than replaces, students' computational thinking and creative agency. This study developed and examined the HAI-MinePy Model, a Human–AI co-creation model for Minecraft Education Python coding. The model positions ChatGPT as a co-creative scaffold that supports MakeCode-to-Python translation, algorithmic brainstorming, concept explanation, and debugging hints, while students remain responsible for problem framing, testing, revision, and final artefact explanation. Grounded in constructionism, computational thinking, scaffolding theory, AI literacy, and human-centred AI, the model was implemented through seven iterative phases: problem framing, prompt formulation, code translation, algorithm co-design, human debugging, code refinement, and reflective evaluation. A mixed-methods design-based research approach was employed with 32 students across eight instructional sessions. Data were collected through computational thinking tests, Python coding tasks, prompt records, debugging logs, Minecraft artefacts, observations, reflection journals, and interviews. Findings showed substantial improvements in students' computational thinking and Python coding performance, particularly in loops, decomposition, algorithmic thinking, debugging, and code explanation. Prompt records indicated that students increasingly used ChatGPT for translation, explanation, and debugging hints rather than direct answer generation. Qualitative findings further revealed that Minecraft functioned as a visible testing environment, while debugging became the central mechanism through which Human–AI co-creation became educationally meaningful. Nevertheless, early tendencies towards AI over-reliance highlighted the need for teacher scaffolding, prompt rules, and process documentation. The study

concludes that the HAI-MinePy Model offers a theoretically grounded and pedagogically practical framework for integrating generative AI into coding education while preserving student agency, creativity, accountability, and responsible AI literacy. It also contributes to emerging scholarship on Human–AI co-creation by demonstrating how learner-centred AI use can be systematically designed, assessed, and refined.

Keywords Human–AI co-creation; Minecraft education; Python coding; computational thinking; AI literacy

Received: 15 June 2026, **Accepted:** 22 August 2026, **Published:** 30 September 2026

Introduction

Generative artificial intelligence has introduced a new pedagogical condition for coding education. Students can now obtain Python code, syntax explanations, algorithmic suggestions, and debugging support almost instantly through conversational AI. This capability is especially relevant in school-based programming contexts where learners often struggle with the transition from block-based coding to text-based programming (Jean et al., 2025; Mladenović et al., 2024). At the same time, this capability raises a central educational concern: when AI can produce code quickly, students may complete programming tasks without developing computational thinking, conceptual understanding, or debugging resilience. Empirical studies have shown that while generative AI can support programming learning, excessive reliance may reduce deeper engagement unless guided by structured pedagogical scaffolds (Zhong et al., 2024; Guo et al., 2026).

Minecraft Education offers a productive environment for addressing this concern because programming is immediately connected to visible action in a dynamic digital world. Python commands can control the Minecraft Agent to move, mine, build, clear space, and create functional structures. This makes abstract programming concepts observable. A loop becomes repeated movement, a sequence becomes an ordered set of Agent actions, and debugging becomes the process of comparing intended outcomes with visible behaviour in the Minecraft world. For young learners, this visibility can reduce the abstraction barrier of Python coding while strengthening the relationship between code, logic, and computational action. Research has shown that Minecraft-based coding environments significantly improve students' understanding of computational concepts such as loops, sequencing, and debugging through interactive and artefact-based learning (Kutay & Oner, 2022).

However, the presence of Minecraft alone does not guarantee deep coding learning, and the presence of ChatGPT alone does not guarantee responsible AI-supported learning. A clear pedagogical model is required to guide how students should interact with AI, how they should test AI-generated suggestions, and how teachers can protect learner agency while still benefiting from AI scaffolding. Previous studies on AI-supported learning emphasise the importance of human–AI co-creation models in maintaining learner autonomy while leveraging AI as a scaffold rather than a replacement for thinking (Dogan, 2025; Romero, 2025).

The study responds to this need by positioning ChatGPT as a co-creative scaffold within Minecraft Education Python coding. In this model, students frame the problem, write prompts, interpret AI suggestions, test code in Minecraft, debug errors, refine the solution, and explain their final artefact. AI contributes explanations, translations, pseudocode, and hints, but the learner remains responsible for judgement, revision, and ownership. This aligns with emerging evidence that structured AI scaffolding

can significantly enhance computational thinking, especially algorithmic thinking and abstraction, when learners remain actively engaged in evaluation and refinement processes (Guo et al., 2026; Liao et al., 2024).

This study focuses on two main objectives. First, it develops the HAI-MinePy Model as a theoretically grounded Human–AI co-creation model for teaching Minecraft Education Python coding. Second, it examines how the model supports computational thinking, particularly sequencing, loops, decomposition, algorithm design, testing, and debugging. These computational thinking components are widely recognised as foundational skills in computer science education and are strongly associated with problem-solving and algorithmic reasoning (Sun et al., 2024; Guss et al., 2024).

Problem Statement

The integration of generative AI into coding education presents a pedagogical paradox. AI can lower barriers to Python learning by providing explanations, translations, and debugging hints. Empirical evidence shows that generative AI can offer real-time scaffolding that enhances students' understanding of programming concepts and supports computational thinking development (Guo et al., 2026; Liao et al., 2024). Yet the same support may weaken learning if students copy AI-generated code without understanding how it works. Research on AI-augmented programming indicates that learners who rely heavily on AI-generated outputs tend to engage less in critical processes such as solution development and refinement, which are essential for deeper learning (Zhong et al., 2024). This problem is particularly important for younger learners who may perceive AI outputs as authoritative and may lack the experience to evaluate syntax, logic, or algorithmic suitability.

The transition from MakeCode blocks to Python in Minecraft Education adds another layer of complexity. Students must move from visual, drag-and-drop representations to textual syntax involving indentation, functions, parameters, loops, and sequence control. The block-to-text transition has been widely identified as a major challenge in programming education, often leading to misconceptions and cognitive overload when not properly scaffolded (Jean et al., 2025; Mladenović et al., 2024). Without support, this transition may overwhelm learners. With unstructured AI support, however, students may bypass the productive struggle that is necessary for computational thinking development. Therefore, the key problem is not whether AI should be used, but how it should be pedagogically structured so that students remain active thinkers and responsible co-creators.

Current literature on AI-supported coding often emphasises productivity, prompt use, or code generation. Less attention has been given to pedagogical models that connect AI interaction with visible testing environments, iterative debugging practices, and computational thinking development. Recent

literature highlights the importance of shifting from passive AI use to human–AI co-creation, where learners actively engage in iterative cycles of generating, evaluating, and refining solutions (Romero, 2025; Dogan, 2025). Furthermore, computational thinking components such as sequencing, loops, and debugging require sustained engagement and deliberate practice to develop effectively, rather than simple code consumption (Sun et al., 2024; Guss et al., 2024). Therefore, this study addresses that gap by pursuing two objectives: (i) To develop a theoretically grounded Human–AI co-creation model for teaching Minecraft Education Python coding, and (ii) To examine how the model supports computational thinking, particularly sequencing, loops, decomposition, algorithm design, testing, and debugging.

Research Objectives

This study was guided by the following research objectives:

- a) To develop and validate the HAI-MinePy Model as a theoretically grounded Human–AI co-creation model for teaching Minecraft Education Python coding (RO1).
- b) To examine how the implementation of the HAI-MinePy Model supports students’ computational thinking, particularly sequencing, loops, conditions, decomposition, algorithmic thinking, testing, debugging, code refinement and code explanation (RO2).

Literature Review

Human–AI Co-Creation in Education

Human–AI co-creation refers to a collaborative process in which humans and AI systems jointly contribute to the production, refinement, or transformation of an artefact, idea, solution, or learning outcome. In educational contexts, co-creation is not simply the delegation of work to AI; rather, it involves reciprocal interaction where learners define goals, interpret AI suggestions, make decisions, evaluate outputs, and iteratively refine their work. Recent research highlights a shift from passive consumption of AI-generated content towards active co-creation, where learners engage in higher-order cognitive processes through iterative dialogue and refinement (Romero, 2025; Dogan, 2025).

In coding education, Human–AI co-creation can occur when students prompt AI tools to suggest code, translate programming logic, identify potential errors, explain syntax, or propose alternative algorithms. However, genuine co-creation requires learners to critically engage with AI outputs through testing, debugging, modification, and justification. Studies on AI-supported programming show that meaningful learning occurs when students actively refine and evaluate AI-generated solutions rather than passively accepting them (Zhong et al., 2024; Guo et al., 2026).

Within this perspective, the human learner remains the primary agent of judgement, while AI functions as a generative and dialogic scaffold. This aligns with scaffolding theories in AI-supported learning, which emphasise that AI should support, not replace, the core cognitive processes such as problem-solving, algorithmic thinking, and debugging (Liao et al., 2024; Shneiderman, 2020; Wood et al., 1976). This distinction is central to the HAI-MinePy Model. The model explicitly rejects the view of ChatGPT as an autonomous code producer and instead positions it as a co-creative coding partner. The student's role is therefore not limited to prompt construction; it extends to problem formulation, algorithmic reasoning, code testing, debugging, ethical evaluation, and ownership of the final artefact. These practices are consistent with core computational thinking competencies that require active engagement and iterative refinement (Sun et al., 2024; Guss et al., 2024).

Minecraft Education as a Constructionist Coding Environment

Minecraft Education is well-suited for constructionist learning because students learn by designing, building, testing, and improving artefacts in a digital world. Constructionism posits that learning is most effective when learners actively create meaningful products that externalise their thinking processes (Papert, 1980; Kutay & Oner, 2022). In Minecraft Education, coding is directly connected to visible and interactive outcomes: an Agent moves, mines, builds, farms, clears areas, or constructs structures based on the code written by students. This visibility reduces abstraction and supports learners in linking programming logic to observable actions, which has been shown to strengthen computational thinking development (Kutay & Oner, 2022; Guss et al., 2024).

For young learners, abstract programming constructs become embodied in spatial and visual actions. For example, a loop is not merely a syntactic structure but becomes repeated Agent movement, while a condition is represented as a decision-making action within the Minecraft environment. Debugging similarly becomes an experiential process, where students observe discrepancies between intended and actual Agent behaviour and iteratively refine their code. Research on computational thinking emphasises that practices such as testing and debugging are essential for developing problem-solving and algorithmic reasoning skills (Sun et al., 2024).

Based on this constructionist affordance, students were required to create Minecraft artefacts through Python coding. While ChatGPT provides support in the form of explanations, code suggestions, and debugging hints, the responsibility for evaluating and refining the code remains with the learner. This aligns with human–AI co-creation principles, where AI functions as a scaffold rather than a substitute for learning, ensuring that students actively engage in testing whether their code produces the intended outcomes in the Minecraft environment (Romero, 2025; Guo et al., 2026).

Python Coding and the Transition from Blocks to Text

Learners often begin programming in block-based environments because they reduce syntax complexity and allow students to focus on logical structures rather than textual correctness. Research shows that block-based programming environments can effectively support early computational thinking by minimising syntax-related errors and enabling learners to concentrate on sequencing, loops, and problem-solving strategies (Mladenović et al., 2024; Guss et al., 2024). However, the transition from block-based coding to Python introduces new challenges. Students must understand indentation, syntax, functions, events, parameters, and textual sequencing, which significantly increases cognitive demands. This block-to-text transition has been widely identified as a critical learning barrier that requires structured scaffolding to prevent misconceptions and cognitive overload (Jean et al., 2025).

In Minecraft Education, this transition is especially important because Python provides a bridge from beginner coding to more formal programming practices. One key pedagogical strategy to support this transition is code translation, where students compare a MakeCode block sequence with its Python equivalent. For example, a “repeat” block can be translated into a for loop, while an Agent movement block can be represented as a Python command (e.g., `agent.move(FORWARD, 1)`). Such translation supports conceptual mapping between visual and textual representations, reinforcing understanding of core programming constructs (Jean et al., 2025; Mladenović et al., 2024).

ChatGPT can further support this process by generating equivalent Python structures and explaining each line in accessible language. Studies indicate that AI tools can facilitate programming comprehension and provide immediate scaffolding through explanations, examples, and debugging suggestions (Guo et al., 2026; Liao et al., 2024). However, translation alone is insufficient. Effective learning requires students to understand why the translated code works, how it may fail, and how it can be adapted to new contexts. Research on AI-supported programming emphasises that deeper learning occurs when learners actively engage in evaluating, modifying, and refining AI-generated code rather than passively accepting it (Zhong et al., 2024). Therefore, this study extended beyond code translation by integrating debugging, testing, and creative application. These practices align with core computational thinking processes—particularly algorithmic reasoning, problem decomposition, and debugging—which require iterative experimentation and reflective practice to develop effectively (Sun et al., 2024).

Computational Thinking and Debugging

Computational thinking involves the ability to formulate problems, decompose tasks, design algorithms, recognise patterns, abstract key features, and systematically test solutions (Wing, 2006). These elements are widely recognised as foundational competencies in computer science education and

are strongly linked to effective problem-solving and programming practices (Guss et al., 2024; Sun et al., 2024). In coding education, debugging is one of the most critical expressions of computational thinking because it requires learners to identify discrepancies between intended and actual outcomes and to iteratively refine their solutions. Research shows that debugging plays a central role in developing algorithmic reasoning and reflective problem-solving skills (Sun et al., 2024).

In Minecraft Education Python coding, debugging may involve syntax errors, indentation errors, incorrect sequencing of Agent actions, misuse of loops, incorrect direction commands, or flawed algorithmic logic. For example, if the Agent moves before destroying a block, it may fail to clear a path. Similarly, incorrect loop indentation may cause the program to behave unexpectedly or fail to execute repeated actions correctly. These errors create valuable learning opportunities, allowing students to reason about code structure, control flow, and program behaviour. Studies on Minecraft-based coding environments show that students frequently engage in testing and debugging while creating artefacts, reinforcing their understanding of computational concepts and practices (Kutay & Oner, 2022).

This study positions debugging as a central rather than peripheral activity in the learning process. Students are encouraged to use ChatGPT as a scaffold by requesting hints, explanations, or partial guidance instead of complete solutions. This approach aligns with research on AI-supported programming, which suggests that meaningful learning occurs when learners actively evaluate and refine AI-generated outputs rather than passively accepting them (Zhong et al., 2024; Liao et al., 2024). Students document errors, test potential corrections, explain their reasoning, and reflect on their learning. This transforms debugging from a purely technical task into a co-creative and metacognitive process. Such practices support deeper engagement with computational thinking by integrating reflection, iteration, and learner agency—key components required for the development of robust programming competence (Guo et al., 2026).

AI Literacy and Responsible AI Use

AI literacy is increasingly important in education because students are not only learning with AI but also learning about AI. AI literacy includes understanding the capabilities and limitations of AI systems, using AI tools effectively, verifying outputs, recognising bias, protecting privacy, and reflecting on human responsibility. Recent research emphasises that AI literacy must go beyond technical use to include critical evaluation and ethical awareness, enabling learners to become responsible and informed users of AI technologies (Romero, 2025; Dogan, 2025).

In this study, AI literacy is embedded directly into coding tasks. Students learn that ChatGPT may produce useful suggestions but may also generate incorrect, incomplete, or contextually

inappropriate code. Studies on generative AI in programming education highlight that AI outputs are not always reliable and require human validation, particularly in learning contexts where conceptual understanding is still developing (Guo et al., 2026; Zhong et al., 2024). Therefore, every AI-generated output must be tested in the Minecraft environment, reviewed critically by students, and revised before acceptance. This process reinforces the idea that AI is a supportive tool rather than an authoritative source.

Students are also required to disclose how AI was used in their work and to explain their own contribution to the final artefact. This aligns with emerging educational practices that promote transparency, accountability, and ethical engagement in AI-supported learning environments (Romero, 2025). By engaging in these practices, learners develop a balanced understanding of AI: they recognise its potential to support creativity and problem-solving while also understanding its limitations.

Furthermore, this approach fosters critical and reflective thinking, ensuring that human reasoning remains central in the learning process. Research on AI-supported computational thinking demonstrates that meaningful learning occurs when students actively evaluate and refine AI-generated outputs, rather than passively accepting them (Liao et al., 2024; Guo et al., 2026). Thus, AI literacy should be viewed as an essential component of coding education, ensuring that learners develop not only programming skills but also the ethical judgement, responsibility, and metacognitive awareness needed in an AI-driven world.

The HAI-MinePy Model

In this study, the HAI-MinePy Model is a Human–AI co-creation model designed for teaching Minecraft Education Python coding. Its central design principle is that AI-generated suggestions must remain provisional until they are tested, debugged, modified, and explained by the learner. The model conceptualises coding as an iterative co-creative process rather than a linear process of receiving solutions from AI (Shneiderman, 2020). This aligns with recent research emphasising that effective AI-supported learning requires active human evaluation and refinement of AI-generated outputs to sustain deeper cognitive engagement (Zhong et al., 2024; Guo et al., 2026).

The model consists of seven interrelated instructional components: human problem framing, prompt formulation, AI-supported code translation, algorithm co-design, human debugging, code refinement, and reflective evaluation, as shown in Figure 1. These components describe how the model functions as a coherent pedagogical system grounded in iterative learning and knowledge construction. Such cyclical processes reflect co-creation frameworks in AI-supported education, where learners engage in cycles of generating, evaluating, and refining outputs (Romero, 2025; Dogan, 2025).

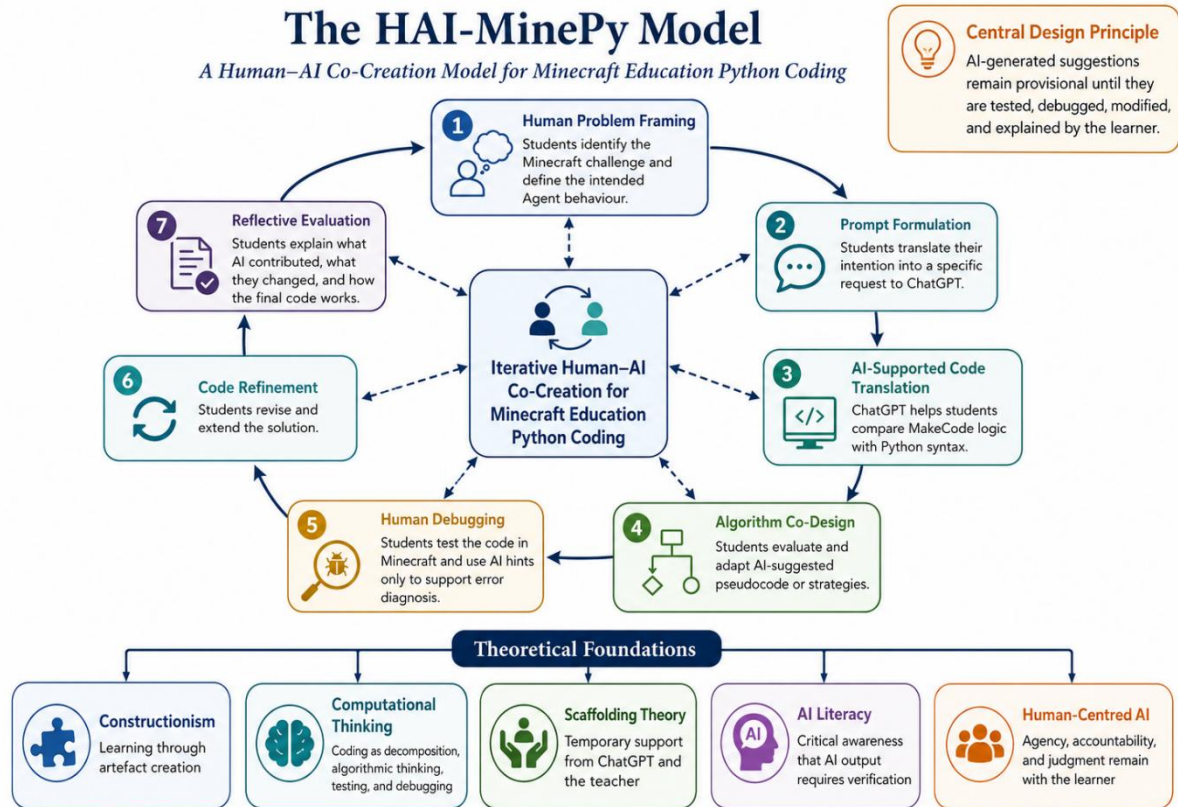


Figure 1. The HAI-MinePy Model

In the human problem framing stage, students identify the Minecraft challenge and define the intended Agent behaviour. In prompt formulation, they translate their intention into structured queries to ChatGPT, reflecting the importance of question formulation in AI-assisted learning (Zhong et al., 2024). During AI-supported code translation, ChatGPT assists students in mapping MakeCode logic to Python syntax, supporting the transition from visual to text-based programming (Jean et al., 2025). In the algorithm co-design stage, students evaluate and adapt AI-generated pseudocode or strategies, ensuring alignment with their intended solution.

In human debugging, students test their code within the Minecraft environment and use AI-generated hints selectively to diagnose and resolve errors. Research consistently highlights debugging as a central process in developing computational thinking and problem-solving skills (Sun et al., 2024; Kutay & Oner, 2022). In the code refinement stage, students iteratively improve and extend their programs, reinforcing algorithmic reasoning and design skills. Finally, in reflective evaluation, students explain how the code works, what AI contributed, and how they modified the solution. This stage supports metacognitive awareness and accountability in AI-supported learning (Liao et al., 2024).

The model is grounded in five theoretical commitments. First, constructionism supports learning through the creation of meaningful artefacts, particularly in interactive environments such as Minecraft (Kutay & Oner, 2022; Papert, 1980). Second, computational thinking provides the cognitive framework for problem solving, including decomposition, sequencing, algorithm design, and debugging (Guss et al., 2024; Sun et al., 2024; Wing, 2006). Third, scaffolding theory explains the role of ChatGPT and teachers as providers of temporary support that guides learners towards independent problem solving (Liao et al., 2024; Wood et al., 1976). Fourth, AI literacy emphasises the need for critical evaluation of AI outputs, ensuring that learners verify, adapt, and take responsibility for their work (Romero, 2025; Guo et al., 2026). Finally, human-centred AI highlights that agency, accountability, and judgement must remain with the learner, positioning AI as a collaborative partner rather than an autonomous problem solver (Shneiderman, 2020; Zhong et al., 2024).

Research Gap and Novelty of the HAI-MinePy Model

Previous Human–AI co-creation frameworks have explained how learners may move from passive consumption of AI outputs towards more active forms of dialogue, evaluation, and co-design. Similarly, AI-assisted programming research has examined code generation, explanations, prompting, and the influence of AI support on programming performance. However, these approaches do not always specify how AI-generated code should be tested, transformed, and documented in a visible game-based programming environment.

Research on Minecraft and game-based programming has shown the educational value of interactive construction, immediate visual feedback, experimentation, and computational thinking. Block-to-text transition studies have also shown the importance of structured bridges between visual programming and Python. Nevertheless, game-based coding models commonly focus on motivation, engagement, programming concepts, or artefact production, whereas the governance of generative-AI participation, learner accountability, and process documentation remains less explicit.

The HAI-MinePy Model of this study aims to address this intersection by combining seven elements within one instructional process: human problem framing, structured prompt formulation, MakeCode-to-Python translation, algorithm co-design, visible Minecraft execution, human-led debugging, and reflective disclosure of AI and student contributions. Its novelty does not lie merely in using ChatGPT together with Minecraft Education. Rather, it lies in positioning debugging as the central accountability mechanism through which provisional AI suggestions are tested, questioned, corrected, and transformed into learner-owned computational solutions. In doing so, the model builds on Jean et al.'s (2025) scaffolded approach to the block-to-Python transition and on Mladenović et al.'s (2024) evidence that a structured MakeCode-to-Python bridge reduces programming misconceptions.

Research Methodology

Research Design

This study employed a mixed-methods design-based research (DBR) approach (Anderson & Shattuck, 2012) because it aimed both to develop a practical pedagogical model and to generate theoretical understanding of Human–AI co-creation in coding education. Design-based research was appropriate because the HAI-MinePy Model was developed, evaluated, implemented and refined within an authentic classroom setting rather than being treated as a fixed intervention. The DBR process was conducted through two linked design and refinement cycles.

During Cycle 1: Initial Design and Expert Refinement, the preliminary HAI-MinePy Model was constructed from five theoretical foundations: constructionism, computational thinking, scaffolding theory, AI literacy and human-centred AI. Seven experts then evaluated the model's theoretical grounding, instructional phases, Human–AI role division, computational thinking alignment, feasibility and supporting instruments. Quantitative Content Validity Index (CVI) results and qualitative comments were used to revise the model. The principal refinements included clearer separation of human and AI roles, stronger prompt-use rules, more systematic debugging documentation, additional teacher-scaffolding guidance and more explicit responsible-AI reflection prompts. Expert validation data were collected to examine the theoretical coherence, pedagogical clarity, Human–AI role division, classroom feasibility, and computational thinking alignment of the HAI-MinePy Model. The expert review process combined quantitative CVI analysis and qualitative feedback to ensure both rigour and depth of evaluation (Lynn, 1986). This approach aligns with best practices in educational model validation, where expert judgement is used to refine instructional frameworks and ensure theoretical alignment (Lynn, 1986; Polit et al., 2007).

During Cycle 2: Classroom Enactment and Implementation Refinement, the revised model was implemented with 32 upper-primary students over eight instructional sessions. Evidence was collected from classroom observations, ChatGPT prompt records, debugging logs, code versions, Minecraft artefacts, student reflections, interviews, computational thinking assessments and Python coding tasks. These data were used to examine how the model operated in practice and to identify further refinements relating to prompt formulation, teacher support, code testing, debugging and learner accountability.

For RO2, student learning data were collected to examine how the model supported computational thinking through Minecraft-based Python coding. Data sources included debugging logs, ChatGPT prompt records, code versions, and reflective explanations. These data sources allowed for analysis of students' engagement in key computational thinking practices such as sequencing, algorithm design, testing, and debugging. Prior research has demonstrated that analysing student artefacts and

iterative coding processes provides valuable insights into computational thinking development and learning progression (Kutay & Oner, 2022; Sun et al., 2024). Additionally, the inclusion of AI interaction data, for example, prompts and responses, aligns with emerging methodologies for studying AI-supported learning processes and human–AI co-creation dynamics (Zhong et al., 2024; Guo et al., 2026). Across the DBR cycles, refinement decisions were based on a combination of expert judgement, observed classroom difficulties, student interactions with ChatGPT, coding errors, debugging behaviour and reflective explanations. Therefore, the final model represented the outcome of iterative design, enactment, evaluation and refinement rather than a single implementation of a predetermined intervention.

Participants

The participants of this study consisted of 32 upper-primary students who had prior exposure to Minecraft Education or block-based programming environments. A purposive selection strategy was used. Students were eligible to participate if they were enrolled in the school club and had prior experience with Minecraft Education or block-based programming. This prior experience was considered important because the learning sequence required students to progress from visual MakeCode representations to Python text-based coding.

Such learners are considered suitable for exploring transitions from visual to text-based coding, as prior studies indicate that foundational experience with block-based programming supports the development of computational thinking and facilitates progression to more advanced coding concepts (Jean et al., 2025; Mladenović et al., 2024). Written parental consent and student assent were obtained before data collection. Participation was voluntary, and students were informed that they could withdraw without penalty. All 32 participants completed both the computational thinking pre-test and post-test and were included in the paired-sample analysis.

The study was conducted in a single educational setting. Therefore, the sample was appropriate for the exploratory and iterative purposes of design-based research but was not intended to be statistically representative of all upper-primary students. The findings should consequently be interpreted as context-specific implementation evidence. Seven experts participated in the model-validation phase. They represented coding education, artificial intelligence in education, educational technology, computational thinking, learning sciences and STEM education. All experts held doctoral qualifications and had between 10 and 18 years of relevant professional experience.

The inclusion of multidisciplinary experts is consistent with best practices in design-based research and instructional model validation, where diverse expertise ensures theoretical rigour,

pedagogical relevance, and practical applicability (Lynn, 1986). Additionally, expert review is widely recognised as a critical step in validating educational frameworks, particularly when examining constructs such as computational thinking, AI integration, and classroom feasibility. Table 1 shows the experts' demographic and professional details.

Table 1. Experts' demographic and professional details

Expert	Area of Expertise	Academic Qualification	Years of Experience	Affiliation Type
Expert 1	Coding Education	PhD	15	University
Expert 2	AI in Education	PhD	12	University
Expert 3	Educational Technology	PhD	10	University
Expert 4	Computational Thinking	PhD	18	University
Expert 5	Coding Education	PhD	14	University
Expert 6	AI & Learning Sciences	PhD	11	University
Expert 7	STEM Education	PhD	13	Teacher Education

Training Module Duration

The module was implemented over eight instructional sessions, each approximately 60 minutes long. The learning sequence followed a progressive structure, moving from simple Agent movement and MakeCode-to-Python code translation to more advanced processes, including algorithm co-design, debugging, project refinement, and reflective explanation. Such staged progression aligns with established instructional approaches in programming education, where scaffolding is gradually reduced as learners develop greater independence and computational thinking competence (Jean et al., 2025; Wood et al., 1976).

ChatGPT was integrated into the sessions in accordance with structured classroom guidelines. Students were permitted to use AI to request explanations, code translations, pseudocode, or debugging hints. However, they were required to test, modify, and explain any AI-supported code before it could be accepted as their own solution. This approach reflects best practices in AI-supported learning, which emphasise that meaningful learning occurs when learners actively engage with, evaluate, and refine AI-generated outputs rather than passively accepting them (Zhong et al., 2024; Liao et al., 2024).

Furthermore, the requirement for students to explain and justify their code aligns with research highlighting the importance of reflection and metacognitive engagement in computational thinking development. Iterative testing, debugging, and explanation are critical processes that enable learners to internalise programming concepts and strengthen problem-solving skills (Sun et al., 2024; Kutay & Oner, 2022). Therefore, the structured use of ChatGPT within the instructional sequence ensured that AI functioned as a scaffold to support learning, while maintaining student agency, accountability, and ownership of the final coding artefacts. Table 2 shows the eight instructional sessions in this study.

Table 2. Eight-session implementation structure

Week	Instructional Focus	Main Activity	Evidence
Week 1	Introduction to Minecraft Python	Agent movement and simple sequences	Pre-test and coding evidence
Week 2	Code Translation	MakeCode blocks to Python	Prompt records and translation worksheets
Week 3	Prompting and Explanation	Asking ChatGPT to explain loops and conditions	Prompt quality and reflection evidence
Week 4	Algorithm Co-Design	Planning mining or building algorithms	Algorithm plans and observation notes
Week 5	Debugging AI Code	Testing and fixing AI-generated code	Debugging logs and code versions
Week 6	Creative Project	Building a Minecraft artefact	Code versions and Minecraft testing evidence
Week 7	Refinement	Improving code and features	Revised code and debugging explanations
Week 8	Reflection and Assessment	Presentation and interview	Post-test, coding task, reflections

Data Collection

Data collection was organised around the two research objectives to ensure alignment between evidence sources and analytical focus. For RO1, data included expert Content Validity Index (CVI) ratings, open-ended expert comments, classroom observations, teacher feedback, ChatGPT prompt records, and student reflection evidence related to the feasibility, coherence, and enactment of the HAI-MinePy Model. The use of expert validation data combined with qualitative feedback aligns with best practices in educational design research, where iterative refinement is supported through both quantitative indices and rich contextual insights (Anderson & Shattuck, 2012; Polit et al., 2007).

For RO2 (computational thinking development), data included computational thinking pre- and post-tests, Python coding tasks, debugging logs, Minecraft artefacts, code versions, student reflections, classroom observations, and interviews. This multi-source data collection approach reflects established practices in computational thinking research, where learner artefacts, iterative code development, and reflective explanations are analysed to capture both process and outcome dimensions of learning (Kutay & Oner, 2022; Sun et al., 2024). Additionally, the inclusion of AI interaction data, such as prompt records and revisions, supports emerging methodologies for examining human–AI co-creation processes and learner engagement with generative AI tools (Zhong et al., 2024; Guo et al., 2026).

Instrument Development and Validation

The computational thinking test comprised 30 items designed to assess six domains: sequencing, loops, conditions, decomposition, algorithmic thinking, and debugging. Five items were allocated to each domain to ensure balanced construct representation. The instrument was researcher-developed based on the conceptualisation of computational thinking proposed by Wing (2006) and the more recent descriptions of computational thinking learning trajectories and debugging practices presented by Guss et al. (2024) and Sun et al. (2024). The contextual design of the assessment tasks was additionally informed by Kutay and Oner’s (2022) work on computational thinking through Minecraft-based coding. The items were modified to reflect the specific demands of Minecraft Education Python programming,

including Agent movement, command sequencing, loop application, error identification, algorithm construction, and code correction. A table of specifications was constructed to map the 30 items to the six computational thinking domains, the eight instructional sessions, and the study's research objectives.

An initial pool of 36 items was prepared before expert validation. Seven experts in coding education, artificial intelligence in education, educational technology, computational thinking, learning sciences, and STEM education evaluated the relevance, clarity, developmental appropriateness, and construct alignment of the items using a four-point relevance scale. Their quantitative ratings were analysed using the Item-level Content Validity Index, Scale-level Content Validity Index average, and modified kappa. Their qualitative comments were used to improve ambiguous wording, simplify technical terminology, clarify task instructions, and strengthen the alignment between individual items and their intended computational thinking domains.

Before the main implementation, the revised computational thinking test was pilot-tested with 30 upper-primary students who had characteristics comparable to those of the main participants but were not included in the final study. The pilot study examined the clarity of the instructions, appropriateness of the language, completion time, preliminary item difficulty, item discrimination, and internal consistency of the instrument. Based on the pilot findings, six items showing weak discrimination, redundancy, or insufficient alignment with their intended domains were removed. In addition, five items were reworded to improve clarity and developmental appropriateness, while four distractors were revised because they were either implausible or insufficiently differentiated from the correct answer. These revisions produced the final balanced 30-item test, comprising five items for each computational thinking domain. The final instrument demonstrated good internal consistency in the main study, with a Cronbach's alpha coefficient of .87.

Students' Python coding performance was assessed using an analytic rubric covering syntax accuracy, logical sequencing, loop application, use of Minecraft Agent commands, debugging and correction, and code explanation. Each dimension was scored on a scale ranging from 0 to 100, with higher scores indicating stronger coding performance. To establish inter-rater reliability, two trained raters independently scored 10 of the 32 coding artefacts, representing 31.25% of the complete dataset. The artefacts were selected through stratified random sampling to include students with low, moderate, and high preliminary coding-performance scores. This procedure ensured that inter-rater agreement was examined across different levels of coding proficiency rather than being based only on high-performing or easily scored artefacts.

Both raters applied the same analytic rubric independently, and inter-rater reliability was calculated before scoring disagreements were discussed. Cohen's κ was .82, indicating almost perfect agreement between the raters (Landis & Koch, 1977). After the reliability coefficient had been established, discrepancies were examined through consensus discussion, and the rubric descriptors were clarified before the remaining artefacts were scored. This procedure reduced subjective scoring variation and strengthened the dependability of the rubric-based findings.

Data Analysis

Quantitative Analysis

Quantitative data were analysed in several stages. The quantitative analysis included descriptive statistics, Content Validity Index (CVI) analysis, pre- and post-test comparisons, effect size estimation, and rubric-based coding performance analysis. The CVI procedure was used to determine expert agreement on the relevance and clarity of the HAI-MinePy Model components and supporting instruments (Lynn, 1986). This approach is widely used in educational research to establish content validity and ensure alignment between theoretical constructs and measurement tools (Polit et al., 2007; Yusoff, 2019). Each expert rated the items using a four-point relevance scale. Item-level CVI (I-CVI) was calculated by dividing the number of experts rating an item as relevant by the total number of experts, while the Scale-level CVI average (S-CVI/Ave) was computed by averaging I-CVI values across all items. Modified kappa values were used to assess agreement beyond chance, providing a more rigorous evaluation of expert consensus (Yusoff, 2019; Polit et al., 2007). Further, qualitative expert comments were analysed thematically to identify areas requiring refinement and to strengthen the model before classroom implementation. This mixed-methods validation approach is consistent with design-based research, where iterative refinement is guided by both measurable agreement and contextual interpretation (Anderson & Shattuck, 2012).

Descriptive statistics were used to calculate students' computational thinking scores and Python coding performance before and after the HAI-MinePy learning sequence. The computational thinking test was scored on a 0–100 scale, with higher scores indicating stronger performance in sequencing, loops, conditions, decomposition, algorithmic thinking, and debugging. The internal consistency of the computational thinking test was examined using Cronbach's alpha. The reliability value was reported to determine whether the test items demonstrated acceptable consistency for measuring students' computational thinking development.

Further, a paired-sample t-test was conducted to examine whether there was a statistically significant difference between students' pre-test and post-test computational thinking scores. This test was appropriate because the same group of students completed the computational thinking test before

and after the intervention. Before conducting the paired-sample t-test, the normality of the difference scores was checked using the Shapiro–Wilk test and inspection of skewness and kurtosis values. Where the normality assumption was reasonably met, the paired-sample t-test was retained. Statistical significance was interpreted at the .05 level. Effect size was reported using Cohen’s d to indicate the magnitude of the pre-post difference for the paired-sample design (Cohen, 1988).

Moreover, the students’ Python coding performance was analysed using a rubric that assessed syntax accuracy, logical sequencing, loop application, use of Agent commands, debugging and correction, and explanation of code. The coding rubric was reviewed by experts to establish content validity and alignment with the targeted computational thinking constructs. To strengthen scoring reliability, two raters independently scored a sample of student coding artefacts, and discrepancies were discussed until agreement was reached. Inter-rater reliability was calculated to determine the consistency of rubric-based scoring. These procedures were used to ensure that the quantitative findings were supported by transparent measurement, reliability, and scoring evidence.

Qualitative Analysis

Qualitative data were analysed using reflexive thematic analysis to explain how students engaged with ChatGPT, Minecraft-based testing, debugging practices, and reflective evaluation throughout the HAI-MinePy learning sequence. Reflexive thematic analysis is widely recognised as a rigorous approach for interpreting patterns of meaning across complex educational datasets, particularly in studies that examine learner experiences and interaction processes (Braun & Clarke, 2021). The qualitative dataset consisted of student reflection journals, semi-structured interview responses, classroom observation notes, ChatGPT prompt records, debugging logs, code versions, and Minecraft artefact evidence. Such multi-modal data sources are commonly used in computational thinking research to capture both process and product dimensions of learning (Kutay & Oner, 2022; Sun et al., 2024).

The analysis began with repeated reading of the qualitative data to achieve familiarisation with students’ learning processes and AI-supported coding behaviours. Initial open coding was then conducted to identify meaningful units related to prompt formulation, AI use, debugging behaviour, learner agency, code refinement, and computational thinking practices. These iterative coding procedures align with established qualitative research practices in educational design research, where meaning emerges through systematic engagement with data (Braun & Clarke, 2021). The initial codes were subsequently compared across data sources and clustered into broader thematic categories, reflecting patterns in how students engaged in human–AI co-creation and problem-solving. These themes were reviewed, refined, and named based on their relevance to the research objectives and their explanatory value in relation to Human–AI co-creation processes (Romero, 2025; Dogan, 2025).

To enhance credibility, the qualitative findings were triangulated across multiple evidence sources. Students' reflection statements were systematically compared with ChatGPT prompt records, debugging logs, code versions, observation notes, and final Minecraft artefacts. This triangulation process ensured that reported experiences were supported by observable coding behaviours and artefact development, which is a recommended strategy for strengthening trustworthiness in qualitative educational research (Morse, 1991). An audit trail was maintained through detailed coding notes, theme development records, and representative evidence excerpts, enhancing the transparency and dependability of the analysis.

Importantly, the qualitative analysis did not treat student reflections as isolated perceptions but as part of a broader process-based dataset. This approach allowed the study to capture how students framed problems, interacted with AI, tested code, debugged errors, refined solutions, and explained their final artefacts. Such process-oriented analysis is increasingly emphasised in studies of AI-supported learning, where meaning is constructed through iterative human–AI interaction rather than through isolated outcomes (Zhong et al., 2024; Guo et al., 2026).

Table 3. Data collection matrix for research objectives, data sources and analysis

Research Objective	Data Sources	Data Analysis	Purpose
RO1: To develop a theoretically grounded Human–AI co-creation model for Minecraft Education Python coding	Expert CVI ratings, open-ended expert feedback, classroom observations, teacher feedback, prompt records, and student reflections	I-CVI, S-CVI/Ave, modified kappa, thematic analysis of expert comments, model feasibility analysis, Human–AI role clarity analysis, teacher scaffolding analysis, and student agency evidence	To examine the content validity, theoretical coherence, pedagogical clarity, and classroom feasibility of the HAI-MinePy Model
RO2: To examine how the HAI-MinePy Model supports students' computational thinking in Python coding	CT pre/post-test, Python coding tasks, debugging logs, code versions, Minecraft artefacts, prompt records, classroom observations, interviews, and student reflections	Descriptive statistics, paired pre/post comparison, effect size estimation, coding rubric analysis, and reflexive thematic analysis of qualitative process evidence	To examine students' computational thinking development, particularly sequencing, loops, decomposition, algorithm design, testing, debugging, code refinement, and code explanation

Ethical Procedures and Responsible AI Use

Ethical procedures were carefully implemented because the study involved upper-primary students, AI-supported learning activities, digital artefacts, prompt records, debugging logs, reflections, and interview data. Before data collection, permission was obtained from the participating school, and written parental consent as well as student assent were secured. Students were informed that participation was voluntary, that they could withdraw at any stage, and that their data would be used solely for research purposes. These practices align with established ethical standards in educational research involving minors, which emphasise informed consent, participant protection, and voluntary participation.

To protect participant confidentiality, all student data were anonymised using identifiers such as S1, S2, and S3. No real names appeared in analysis, reporting, tables, or figures. Minecraft artefacts, reflection journals, interview responses, debugging logs, and prompt records were securely stored and accessed only by the researcher. Any identifying details were removed before analysis. Such procedures are consistent with widely accepted ethical practices in educational data management, ensuring privacy, confidentiality, and secure handling of sensitive information.

Specific safeguards were also applied to the use of ChatGPT during coding activities. Students did not use personal accounts and interacted with AI only under teacher supervision within a controlled classroom environment. They were explicitly instructed not to enter any personal or sensitive information into ChatGPT. The AI tool was used exclusively for learning purposes, including code translation, syntax explanation, algorithm exploration, and debugging support. These precautions reflect emerging guidelines for responsible AI use in education, which emphasise privacy protection, controlled access, and pedagogically guided interaction.

Students were also guided to understand that AI-generated responses were not authoritative or final. All AI-supported outputs had to be tested in Minecraft Education, evaluated against intended outcomes, modified where necessary, and explained by the student before inclusion in final artefacts. This requirement aligns with principles of AI literacy, which emphasise critical evaluation, verification, and responsible use of AI outputs (Romero, 2025; Liao et al., 2024; Long & Magerko, 2020). It also ensured that ChatGPT functioned as a scaffold to support learning rather than as a substitute for computational thinking.

Prompt records and AI interaction data were treated as part of the research dataset because they provided evidence of students' problem formulation, AI engagement, and iterative refinement processes. These records were anonymised and analysed only in aggregated form or through non-identifiable excerpts. The study therefore adhered to key ethical principles, including informed consent, assent, confidentiality, data protection, responsible AI use, and teacher-supervised interaction. Furthermore, by embedding AI literacy into the learning process, the study ensured that ethical awareness and accountability were integral to students' engagement with AI-supported coding (Long & Magerko, 2020).

Findings

The findings are reported according to the two research objectives. The first two subsections address RO1 by presenting the expert validation findings and the classroom enactment of the HAI-MinePy Model. The next four subsections address RO2 by reporting students' computational thinking gains,

Python coding performance, debugging performance, and qualitative learning processes. The final subsection integrates the quantitative and qualitative findings to explain how Human–AI co-creation supported computational thinking development in Minecraft Education Python coding.

Expert Validation of the HAI-MinePy Model

Expert content validation was conducted to address RO1 before the classroom implementation. Seven experts reviewed the HAI-MinePy Model and its supporting learning evidence using a structured validation form. The experts evaluated the relevance and clarity of the model components, Human–AI role division, AI-use rules, teacher scaffolding requirements, computational thinking alignment, debugging documentation, and reflection prompts. The CVI findings in Table 4 showed strong expert agreement across the reviewed components, indicating that the model was suitable for use as a Human–AI co-creation framework in Minecraft Education Python coding.

Table 4. CVI findings for expert validation of the HAI-MinePy Model

Validated component	Experts rating (n=7)	I-CVI	Modified kappa	Decision
Theoretical grounding of the model	7	1.00	1.00	Retained
Clarity of Human–AI co-creation roles	7	1.00	1.00	Retained
Suitability of the seven instructional phases	7	1.00	1.00	Retained
AI-supported code translation component	6	0.86	0.85	Retained with minor refinement
Algorithm co-design component	7	1.00	1.00	Retained
Human debugging component	7	1.00	1.00	Retained
Reflective evaluation and responsible AI use	6	0.86	0.85	Retained with minor refinement
Alignment with computational thinking constructs	7	1.00	1.00	Retained
Feasibility for classroom implementation	6	0.86	0.85	Retained with minor refinement
Supporting instruments and process documentation	6	0.86	0.85	Retained with minor refinement

In Table 4, the I-CVI values ranged from .86 to 1.00. Six components achieved unanimous expert agreement, whereas four components were rated as relevant by six of the seven experts. The calculated S-CVI/Ave was .94, indicating excellent overall content validity. Modified kappa values ranged from .85 to 1.00, indicating excellent agreement beyond chance. Components with I-CVI values of .86 were retained because they met the acceptable criterion for a seven-member panel (I-CVI $\geq$.78; Lynn, 1986; Polit et al., 2007); however, the accompanying qualitative comments were used to improve their wording, classroom guidance, and documentation procedures. These results provided quantitative evidence that the model was sufficiently relevant and clear for classroom implementation. The experts' qualitative feedback and the resulting refinements are summarised in Table 5.

Table 5. Expert qualitative feedback and model refinements

Expert feedback theme	Summary of expert feedback	Model refinement made
Clarify Human–AI role division	Experts recommended making the student role more visible so that ChatGPT would not appear to be the main producer of code.	The model description was revised to distinguish student responsibility for problem framing, testing, debugging, modification, and explanation from AI support for translation, hints, and suggestions.

Strengthen prompt rules	Experts cautioned that students might ask ChatGPT for complete answers during early coding tasks.	Prompt rules were added to encourage requests for explanations, pseudocode, and debugging hints rather than direct final code.
Emphasise debugging as learning evidence	Experts suggested that debugging should be treated as the central evidence of computational thinking rather than a minor correction activity.	Debugging logs were strengthened to require students to record intended outcomes, observed errors, possible causes, AI hints, corrections, and retesting results.
Improve teacher scaffolding	Experts noted that younger learners would require guidance when interpreting AI-generated Python syntax.	Teacher facilitation notes were added for prompt modelling, code explanation, and gradual release of support.
Increase responsible AI awareness	Experts recommended clearer attention to verification, transparency, and accountability in AI-supported coding.	Reflection prompts were revised to require students to state what ChatGPT suggested, what they changed, and how they verified the final code.
Align CT indicators with model phases	Experts requested a clearer mapping between model activities and computational thinking constructs.	Sequencing, loops, decomposition, algorithmic thinking, testing, and debugging were explicitly mapped to the relevant HAI-MinePy phases.

Implementation of the HAI-MinePy Model

The findings suggested that the HAI-MinePy Model functioned as a feasible Human–AI co-creation structure for Minecraft Education Python coding. Students were able to engage with the model as an iterative process involving problem framing, prompting, code translation, algorithm planning, testing, debugging, refinement, and reflection. Importantly, the model did not operate as a rigid linear sequence. Students frequently moved back and forth between ChatGPT interaction, Minecraft testing, and code revision. This iterative movement became the defining feature of the model.

The strongest contribution of the model was its clear separation of AI support from human accountability. ChatGPT was used to generate possibilities, but students were required to make decisions, test outputs, and explain revisions. This design prevented the learning process from collapsing into simple code outsourcing. Classroom evidence showed that students were most agentic when they treated AI output as a draft to be interrogated rather than as a final answer.

Teacher scaffolding remained important throughout implementation. Students initially needed support in writing specific prompts, interpreting Python syntax, and diagnosing errors. Over time, however, they became more capable of asking ChatGPT for explanations and hints instead of complete code. This change suggested that the model supported gradual movement from dependence to more regulated co-creation. Table 6 shows the implementation findings for the HAI-MinePy Model in this study.

Table 6. Implementation findings for the HAI-MinePy Model

Model component	Human role	AI role	Observed implementation
Problem framing	Defines Minecraft goal and intended Agent behaviour	No dominant role; may clarify possibilities if asked	Students described intended outcomes such as mining, clearing, and building tasks
Prompt formulation	Translates intention into a clear request	Responds to the student prompt	Prompt quality improved with teacher scaffolding
Code translation	Compares MakeCode logic with Python syntax	Explains equivalent Python structures	Students became more confident mapping blocks to Python commands
Algorithm co-design	Selects, adapts, and justifies the strategy	Suggests pseudocode or alternative steps	Students increasingly used sequences and loops to plan Agent behaviour

Human debugging	Tests the code, identifies mismatches, and decides on corrections	Provides hints or explanations	Debugging became more systematic after repeated practice
Code refinement	Modifies code and adds features	Suggests improvements when requested	Stronger students extended AI suggestions creatively
Reflective evaluation	Explains code, AI contribution, and personal modification	May support reflective questioning	Students identified what they changed and why

Computational Thinking Gains

The quantitative findings showed substantial improvement in students' computational thinking after the HAI-MinePy module, as shown in Table 7. The overall computational thinking mean increased from 56.84 before the intervention to 78.41 after the intervention. The largest gains were recorded in loops, algorithmic thinking, and decomposition. The findings also suggest that students demonstrated measurable learning gains after participating in the HAI-MinePy learning sequence. However, because the study used a small-scale DBR design without a comparison group, the gains should be interpreted as implementation-based evidence.

In this study, the improvement was not interpreted as an effect of ChatGPT alone. Instead, the learning gains were explained by the interaction between AI-supported explanation, visible Minecraft testing, teacher scaffolding, and human-led debugging. Qualitative evidence suggested that students who repeatedly tested and modified AI-supported code demonstrated stronger understanding than those who initially copied code without explanation, although this subgroup difference was not tested statistically.

Table 7. Pre- and post-test scores for computational thinking

CT domain	Pre-test mean	Pre-test SD	Post-test mean	Post-test SD	Mean gain
Sequencing	62.19	11.42	82.34	8.76	20.15
Loops	51.56	13.85	77.03	10.14	25.47
Conditions	49.38	12.91	70.31	11.63	20.93
Decomposition	58.75	10.68	79.69	9.21	20.94
Algorithmic thinking	55.63	12.17	80.47	8.94	24.84
Debugging	63.52	10.95	81.25	8.38	17.73
Overall CT score	56.84	9.87	78.41	7.92	21.57

The computational thinking test was scored on a 0–100 scale. Higher scores indicated stronger computational thinking performance across sequencing, loops, conditions, decomposition, algorithmic thinking, and debugging. The same 32 students completed both the pre-test and post-test; therefore, a paired-sample t-test was used to compare students' computational thinking scores before and after the HAI-MinePy learning sequence. The normality of the difference scores was examined before the paired-sample analysis. The assumption was considered acceptable based on the Shapiro–Wilk test and inspection of skewness and kurtosis values.

The students' mean computational thinking score increased from 56.84 (SD = 9.87) at pre-test to 78.41 (SD = 7.92) at post-test. The mean paired difference was 21.57 points (SD of the difference = 10.68, SE = 1.89). The analysis indicated a statistically significant improvement in computational thinking performance, $t(31) = 11.42$, $p < .001$, 95% CI [17.72, 25.42]. The standardised mean difference, calculated as the mean difference divided by the standard deviation of the differences, was large, Cohen's $d = 2.02$ (Cohen, 1988). Although the magnitude of the pre-post difference was substantial, the findings should be interpreted cautiously because the study did not include a control or comparison group. Accordingly, the results represent implementation-based evidence associated with participation in the HAI-MinePy learning sequence rather than causal evidence that the model alone produced the observed improvement.

Table 8. Paired-sample comparison of overall computational thinking scores

Measure	n	Mean	SD	Mean Difference	95% CI for difference	df	t	p	Effect Size
Pre-test CT score	32	56.84	9.87	21.57	[17.72, 25.42]	31	11.42	< .001	Cohen's $d = 2.02$
Post-test CT score	32	78.41	7.92						

These results provide evidence of substantial computational thinking development following participation in the HAI-MinePy learning sequence. Nevertheless, the single-group design limits causal interpretation of the observed improvement.

Python Coding Performance

The Python coding task was assessed using a rubric aligned with the computational thinking constructs targeted in the HAI-MinePy Model. The rubric assessed syntax accuracy, logical sequencing, use of loops, use of Minecraft Agent commands, debugging and correction, and explanation of code. Each dimension was converted to a score ranging from 0 to 100, with higher scores indicating stronger coding performance. Inter-rater reliability for the rubric (Cohen's $\kappa = .82$) is reported in the Instrument Development and Validation section.

The rubric-based analysis showed that students improved from a baseline mean of 52.66 (SD = 10.95) to a final task mean of 80.94 (SD = 8.21), as shown in Table 9. The largest gain was found in the explanation of code, followed by the use of loops, syntax accuracy, and debugging and correction. These improvements suggest that students were not only producing working Minecraft artefacts but were increasingly able to explain how their Python code controlled Agent behaviour. This finding is important because code explanation, debugging, and modification provided evidence that students were developing computational thinking rather than merely copying AI-generated code.

Table 9. Python coding performance before and after the intervention

Coding dimension	Baseline mean	Baseline SD	Final task mean	Final task SD	Mean gain
Syntax accuracy	48.44	14.36	76.88	10.79	28.44
Logical sequencing	59.06	12.41	84.38	8.67	25.32
Use of loops	46.88	15.27	78.13	10.22	31.25
Use of Agent commands	63.75	11.38	86.25	7.94	22.50
Debugging and correction	51.25	13.72	79.69	9.81	28.44
Explanation of code	46.56	14.18	80.31	9.36	33.75
Overall coding performance	52.66	10.95	80.94	8.21	28.28

Debugging Logs of Students

Table 10 summarises the types and frequencies of the 174 errors documented in students' debugging logs. Sequence errors were the most frequent, followed by indentation, loop and syntax errors. Table 11 presents changes in students' debugging performance across the early, middle and final instructional sessions. These errors were pedagogically valuable because they required students to compare their intended Minecraft outcome with the actual behaviour of the Agent and the structure of the Python code. The findings in Table 11 suggest that students' debugging performance improved across the module. In the early sessions, many students could state that the code did not work, but could not explain the cause. By the later sessions, they increasingly identified error types, corrected code successfully, tested revised code, and requested hints rather than complete answers. This shift suggests movement from surface-level correction to computational reasoning.

Table 10. Types of errors identified in debugging logs

Error type	Frequency	Percentage
Syntax errors	24	13.79%
Indentation errors	31	17.82%
Sequence errors	36	20.69%
Loop errors	29	16.67%
Direction or movement errors	22	12.64%
Logic errors	18	10.34%
Minecraft execution errors	14	8.05%
Total	174	100.00%

Table 11. Student debugging performance across the module

Debugging indicator	Sessions 1-2	Sessions 3-5	Sessions 6-8
Errors correctly identified	41.18%	68.42%	84.21%
Errors successfully corrected	35.29%	63.16%	81.58%
Students explained the cause of the error	29.41%	57.89%	78.95%
Students tested the revised code	52.94%	76.32%	92.11%
Students requested hints rather than answers	23.53%	55.26%	73.68%

Qualitative Themes from Reflections, Debugging Logs, Prompt Records, and Interviews

The qualitative analysis generated five themes that explained how students experienced Human–AI co-creation during Minecraft Education Python coding. These were: (1) from answer-seeking to hint-seeking, (2) Minecraft as a visible debugging environment, (3) debugging as the core mechanism of Human–AI co-creation, (4) emerging learner agency through prompt refinement and code explanation, and (5) early AI over-reliance and the need for teacher scaffolding. These themes were derived from student reflection journals, interview responses, classroom observation notes, ChatGPT prompt records,

debugging logs, code versions, and Minecraft artefact evidence. Together, the themes showed that students' computational thinking development was not produced by AI assistance alone, but through iterative cycles of prompting, testing, debugging, modifying, and explaining code.

Theme 1: From Answer-Seeking to Hint-Seeking

At the beginning of the intervention, several students tended to use ChatGPT as a direct answer provider. Their early prompts often asked for complete Python code or immediate solutions to Minecraft tasks. However, as the instructional sessions progressed, students gradually shifted towards more specific and learning-oriented prompts. Instead of asking ChatGPT to complete the whole task, they increasingly requested explanations, debugging hints, or clarification of specific Python commands. This shift indicated an emerging understanding that AI should support their thinking rather than replace it. The prompt records showed that students became more intentional in defining problems, describing errors, and asking for partial guidance. This change reflected the development of prompt literacy as a form of learner agency within the HAI-MinePy Model. Representative prompt records and student reflections illustrated this shift:

“At first, I asked ChatGPT to give me the full code because I wanted the Agent to build quickly. Later, I learned that I should ask why the code did not work, so I could fix it myself.”

(Student reflection, S4)

“Explain why my loop only runs one time. Do not give me the full answer. Give me a hint.”

(Prompt record, S11)

“I changed my question to ChatGPT because my first question was too general. When I told ChatGPT what the Agent was doing wrong, the answer became easier to understand.”

(Interview excerpt, S7)

These excerpts suggest that students gradually moved from using ChatGPT for answer generation towards using it as a scaffold for understanding and debugging. This change is important because it shows that prompt formulation became part of the learning process rather than a shortcut to task completion.

Theme 2: Minecraft as a Visible Debugging Environment

The second theme showed that Minecraft Education functioned as a visible testing environment that helped students connect Python code with observable outcomes. Students were able to see immediately whether the Agent moved correctly, repeated actions through loops, cleared blocks, built structures, or failed to complete the intended task. This visual feedback made debugging more concrete because errors were not only shown as syntax problems but also appeared as mismatches between intended and actual Agent behaviour. For example, incorrect sequencing caused the Agent to move before completing an action, while incorrect loop structures caused repeated actions to fail or behave unexpectedly. The visibility of Minecraft, therefore, supported students in identifying, explaining, and correcting errors. The following excerpts show how students used Minecraft's visual feedback to recognise coding errors:

“When I ran the code, the Agent moved forward but did not place the blocks. I knew the order was wrong because I could see the Agent walking before building.”

(Debugging log, S9)

“Minecraft helped me see the mistake. The code looked correct, but the Agent kept digging in the wrong direction. After I watched the Agent, I changed the turn command.”

(Student reflection, S15)

“The student tested the same code three times and observed that the Agent repeated the movement but skipped the block placement. The student then checked the loop structure and changed the position of the place command.”

(Classroom observation note, O3)

These excerpts indicate that Minecraft acted as more than a coding platform. It became a visible diagnostic space where students could compare intended outcomes with actual Agent behaviour. This helped students understand debugging as a process of observation, reasoning, and revision.

Theme 3: Debugging as the Core Mechanism of Human–AI Co-Creation

The third theme indicated that debugging became the central mechanism through which Human–AI co-creation became educationally meaningful. Students did not simply accept AI-generated code as final. Instead, they tested the suggested code, identified errors, compared outcomes with their intended design, and revised the code through repeated attempts. Debugging logs showed that students documented syntax errors, indentation problems, incorrect Agent movement, loop errors, and logic

issues. In many cases, ChatGPT provided hints, but the students still needed to decide whether the suggestions were suitable and how the code should be modified. This finding suggests that co-creation occurred not when AI generated an answer, but when students evaluated, repaired, and transformed AI-supported code through human judgement. The following excerpts illustrate the centrality of debugging:

“ChatGPT gave me a code, but it did not work in Minecraft because the Agent moved too far. I changed the number in the loop and tested again. After three tries, the Agent built the wall correctly.”

(Debugging log, S2)

“The AI helped me find the possible mistake, but I still needed to try the code in Minecraft. Sometimes the AI answer was not exactly right for my world, so I had to change it.”

(Interview excerpt, S18)

“Error found: indentation problem after for loop. Action taken: checked the spacing, asked ChatGPT what indentation means, corrected the line, and tested again.”

(Debugging log, S6)

“I did not just copy the code. I used the AI suggestion, but I changed the movement steps because my house size was different.”

(Student reflection, S21)

These excerpts show that Human–AI co-creation was enacted through testing, debugging, and adaptation. The students’ role was not passive. They evaluated AI-generated suggestions against the Minecraft task requirements and revised the code based on visible outcomes.

Theme 4: Emerging Learner Agency through Prompt Refinement and Code Explanation

The fourth theme showed that students developed greater ownership of their coding artefacts when they were required to explain their final code and distinguish between AI contributions and their own modifications. Reflection journals and interviews indicated that students became more aware of their role in problem framing, prompt writing, testing, and refinement. Code explanation tasks encouraged students to move beyond producing a working artefact towards understanding why the code worked. This process strengthened metacognitive awareness because students had to describe what they asked ChatGPT, what they changed, what errors they fixed, and how the final Minecraft artefact was

produced. Learner agency was therefore observed through students' ability to justify their decisions and explain the logic of their code. The following excerpts show evidence of learner agency:

"My final code is different from ChatGPT's code. ChatGPT suggested using a repeat loop, but I changed the number of repetitions because I wanted the bridge to be longer."

(Student reflection, S13)

"I can explain this part. The loop repeats the Agent movement five times, so I do not need to write the same command again."

(Interview excerpt, S5)

"I asked ChatGPT what the error meant, but I decided where to put the command because I knew what I wanted the Agent to build."

(Student reflection, S19)

"The student was able to identify which part of the code was suggested by ChatGPT and which part was modified independently. The student explained that the loop was retained, but the movement direction and block type were changed to match the final artefact."

(Classroom observation note, O5)

These excerpts suggest that students began to view themselves as authors and evaluators of their coding artefacts. AI supported the process, but students retained responsibility for design decisions, code modification, and final explanation.

Theme 5: Early AI Over-Reliance and the Need for Teacher Scaffolding

The final theme highlighted the risk of early AI over-reliance. Some students initially treated ChatGPT responses as correct without sufficient testing or explanation. This created a risk that students could copy AI-generated code without understanding the underlying computational logic. However, teacher scaffolding, prompt rules, debugging logs, and reflection tasks helped reduce this tendency. Students were reminded that AI-generated suggestions were provisional and had to be tested, modified, and explained before being accepted. This finding reinforces the importance of structured classroom guidance in AI-supported coding education. Within the HAI-MinePy Model, responsible AI use requires not only access to ChatGPT but also clear pedagogical rules, process documentation, and

teacher facilitation. The following excerpts show both the risk of over-reliance and the value of scaffolding:

“At first, I thought ChatGPT’s answer must be correct. But when I pasted the code, it had an error. The teacher told us we must test and explain before using it.”

(Student reflection, S8)

“I copied the code too fast and did not know why it failed. After writing the debugging log, I found that I did not understand the loop.”

(Debugging log, S16)

“The teacher reminded students that AI-generated code was only a suggestion. Students were asked to highlight the parts they changed and explain why the changes were needed.”

(Classroom observation note, O2)

“Now I know I cannot only ask ChatGPT to finish the task. I need to ask, test, fix, and explain.”

(Interview excerpt, S23)

These excerpts show that teacher scaffolding played an important role in preventing passive AI use. The requirement to document prompts, debugging steps, code changes, and final explanations helped students treat ChatGPT as a learning scaffold rather than a replacement for their own thinking. In summary, the qualitative findings support the view that the HAI-MinePy Model functioned as a process-based Human–AI co-creation model. Students’ learning emerged through the interaction between AI-supported explanation, Minecraft-based testing, human debugging, code refinement, and reflective evaluation. The themes further suggest that computational thinking was strengthened when students used AI as a scaffold for reasoning, rather than as a substitute for problem-solving.

Integrated Findings Across Quantitative and Qualitative Data

The findings of this study support both research objectives. For RO1, the HAI-MinePy Model was judged content-valid by the expert panel and proved feasible in classroom enactment as a Human–AI co-creation model for Minecraft Education Python coding. For RO2, the model appeared to support students’ computational thinking development by transforming AI-supported coding into an iterative process of problem framing, prompting, testing, debugging, refinement, and reflection. Computational

thinking was supported through a triadic mechanism: (i) AI-assisted explanation, (ii) Minecraft-based execution, and (iii) human-led debugging. The mechanism operated as follows: AI helped students access possible solutions, Minecraft revealed whether those solutions worked, and debugging required students to explain, revise, and retest. This mechanism strengthened CT because students had to coordinate intention, algorithm, code, and outcome.

Discussion

The findings of this study provide focused support for the HAI-MinePy Model as a Human–AI co-creation model for the development of computational thinking in Minecraft Education Python coding. Generative AI can support coding education when its role is carefully designed as a scaffold for human reasoning, and computational thinking develops most strongly when AI suggestions are tested, debugged, and refined in a visible programming environment. This aligns with recent research showing that AI-supported programming enhances learning outcomes most effectively when learners actively evaluate and refine AI-generated outputs rather than passively consume them (Zhong et al., 2024; Guo et al., 2026).

The study contributes a process-based model of Human–AI co-creation that was strengthened through expert content validation. The CVI findings provided evidence that the model components, Human–AI role division, pedagogical phases, debugging emphasis, and computational thinking alignment were judged relevant and clear by the expert panel. This reflects established practices in educational model validation, where expert feedback is used to ensure instructional coherence and theoretical alignment (Guss et al., 2024). The qualitative expert feedback was equally important because it moved the model beyond conceptual description into a more accountable classroom framework. Expert comments led to stronger prompt rules, clearer teacher scaffolding, more explicit debugging logs, and reflection prompts that required students to disclose, verify, and explain AI-supported code. Such iterative refinement is a key characteristic of design-based research, where models evolve through cycles of expert feedback and classroom enactment (Romero, 2025; Dogan, 2025).

The study suggests that computational thinking is strengthened through the interaction of three elements: (i) AI-supported explanation, (ii) Minecraft-based testing, and (iii) human-led debugging. AI enabled access to explanations and alternative algorithmic strategies, while Minecraft provided a concrete and visual environment where code outcomes could be observed. The importance of such interactive environments is supported by research demonstrating that Minecraft-based coding enhances understanding of computational concepts such as loops, sequencing, and debugging through artefact creation and testing (Kutay & Oner, 2022). Debugging required students to connect intended outcomes,

program logic, and observed behaviour, which is a central mechanism for developing computational thinking skills (Sun et al., 2024).

The strong gains in loops, algorithmic thinking, and decomposition are theoretically meaningful. These domains are often difficult for novice learners because they require abstraction and procedural reasoning. However, in Minecraft Education, loops became repeated Agent actions, decomposition involved breaking down tasks into smaller executable steps, and algorithmic thinking emerged through planning structured sequences of actions. This transformation of abstract constructs into visible, manipulable actions supports constructionist learning principles and has been shown to improve computational thinking outcomes (Kutay & Oner, 2022; Guss et al., 2024; Papert, 1980).

Debugging emerged as the epistemic centre of the model. It transformed AI use from answer acceptance into critical inquiry. When students tested AI-supported code and observed failure, they were required to diagnose whether issues stemmed from sequence errors, indentation, loop structure, direction commands, or logic. This aligns with evidence that structured debugging activities significantly enhance computational thinking and problem-solving abilities (Sun et al., 2024). Asking ChatGPT for hints instead of complete answers further supported learning because it preserved the learner's responsibility to reason through errors. Studies on AI-supported programming suggest that partial guidance and scaffolding promote deeper engagement than full solution provision (Liao et al., 2024; Zhong et al., 2024).

The findings also highlight the importance of prompt literacy, which should not be understood merely as a technical skill. Within the HAI-MinePy Model, prompt literacy became a form of learner agency, as students needed to define goals, specify constraints, request explanations, and determine when to seek hints instead of solutions. This reflects emerging perspectives that position human–AI interaction as a co-creative process requiring critical decision-making and iterative refinement (Romero, 2025). Prompting was therefore integrated into computational thinking processes such as decomposition, abstraction, and debugging.

A critical implication is that generative AI must be embedded within structured pedagogical accountability mechanisms. Without guidance, students may copy AI-generated code without understanding it. With structured rules, debugging logs, code-version tracking, and reflective explanations, students learn to treat AI output as provisional and subject to verification. This aligns with research emphasising the need for AI literacy and responsible use in education, ensuring that human reasoning and accountability remain central (Guo et al., 2026; Romero, 2025).

Therefore, this study contributes to Human–AI co-creation theory by demonstrating that co-creation is not defined by AI contribution alone, but by the extent to which learners transform AI outputs through intention, testing, debugging, and explanation. In coding education, co-creation should be evaluated through process evidence, including problem framing, AI interaction, debugging practices, code revision, and explanation of logic. This aligns with recent work highlighting that meaningful learning in AI-supported environments emerges from iterative human–AI interaction rather than one-way assistance (Zhong et al., 2024).

In terms of pedagogical implications, the findings suggest that teachers should position ChatGPT as a scaffold for explanation, translation, brainstorming, and hints rather than as a code generator. Minecraft Education tasks should emphasise visible testing and repeated debugging cycles. Students should maintain prompt records and code versions to make their reasoning processes transparent. Reflection tasks should require students to distinguish between AI contributions and their own modifications. These practices are consistent with research emphasising transparency, reflection, and active learner engagement in AI-supported learning (Liao et al., 2024; Romero, 2025).

Despite these implications, the study remains bound by its implementation context. The findings were derived from a limited classroom intervention, and the effectiveness of the model may vary across learner age groups, prior coding experience, teacher expertise, AI accessibility, and school infrastructure. Future research should validate the model through larger samples, comparative designs, longitudinal studies, and diverse educational contexts. Furthermore, future studies should examine the role of teacher facilitation in balancing AI support with student agency, as this balance is critical for sustaining meaningful Human–AI co-creation in learning environments.

Limitations and Future Research

Several limitations should be considered when interpreting the findings. First, the study involved only 32 upper-primary students from one educational setting. The sample was appropriate for an exploratory design-based implementation, but limits the transferability of the findings to schools with different student populations, teacher expertise, technological infrastructure, or levels of prior coding experience. Second, this study used a single-group pre-test–post-test design without a comparison group. Therefore, the observed improvements cannot be attributed exclusively to the HAI-MinePy Model. Repeated assessment, teacher support, increased Minecraft familiarity, and normal learning progression may also have contributed to the results. Third, the researcher was involved in the design and classroom implementation of the model. This involvement provided a detailed contextual understanding but may have influenced classroom observations and qualitative interpretation. The use of multiple data sources and an audit trail reduced, but did not eliminate, this possibility.

Fourth, students entered the study with prior exposure to Minecraft Education or block-based coding. The results may differ for complete beginners or students with more advanced Python experience. The model may also operate differently when implemented by teachers who have varying levels of confidence in Python, Minecraft Education, or generative AI. Fifth, the study examined an eight-session learning sequence. It did not determine whether the improvement in computational thinking, responsible AI use, or prompt behaviour was sustained over a longer period or transferred to programming environments outside Minecraft Education.

Addressing the limitations, future research should test the model with larger and more diverse samples, multiple schools, and comparison groups. A quasi-experimental or randomised design could compare HAI-MinePy with Minecraft coding without AI support, unstructured ChatGPT-assisted coding, or conventional Python instruction. Longitudinal studies should examine whether students continue to test and verify AI-generated code after explicit teacher scaffolding is reduced. Future research should also investigate teacher implementation fidelity, differences between novice and experienced coders, and the relationship among prompt quality, debugging behaviour, code understanding, and computational thinking outcomes.

Conclusion

This study addressed two objectives: the development of a theoretically grounded Human–AI co-creation framework for Minecraft Education Python coding and the examination of its role in supporting computational thinking. The findings demonstrate that generative AI is pedagogically valuable not as a substitute for student work but as a structured scaffold that supports reasoning, reflection, and iterative problem-solving. Expert validation confirmed the model’s clarity, coherence, and alignment with computational thinking, while qualitative feedback strengthened its practical classroom implementation, particularly in relation to prompting, debugging, and learner accountability. The findings suggest that computational thinking develops when AI-supported explanations are combined with visible testing in Minecraft and systematic human-led debugging. In this context, debugging emerged as the central learning mechanism, transforming AI-generated outputs into opportunities for inquiry rather than passive acceptance. Gains in loops, algorithmic thinking, and decomposition suggest that abstract programming concepts can be internalised when learners actively test, modify, and explain code. The study contributes a process-based understanding of Human–AI co-creation, where the value of learning lies in how students engage with AI through problem framing, evaluation, debugging, and reflection. Therefore, the HAI-MinePy Model of this study offers a pedagogically robust approach for integrating AI into coding education while preserving learner agency, accountability, and conceptual understanding.

References

- [1] Anderson, T., & Shattuck, J. (2012). Design-based research: A decade of progress in education research? *Educational Researcher*, 41(1), 16–25. <https://doi.org/10.3102/0013189X11428813>
- [2] Braun, V., & Clarke, V. (2021). One size fits all? What counts as quality practice in (reflexive) thematic analysis? *Qualitative Research in Psychology*, 18(3), 328–352. <https://doi.org/10.1080/14780887.2020.1769238>
- [3] Cohen, J. (1988). *Statistical power analysis for the behavioral sciences (2nd ed.)*. Lawrence Erlbaum Associates. <https://doi.org/10.4324/9780203771587>
- [4] Dogan, S. (2025). A human-AI hybrid co-design model: Planning effective instruction with ChatGPT. *Asian Journal of Distance Education*, 20(1), 94–119. <https://doi.org/10.5281/zenodo.15212405>
- [5] Guo, R., Li, G., Miao, H., Pi, Z., & Xie, L. (2026). Impact of generative AI-assisted programming on the computational thinking of high school students. *Frontiers in Psychology*, 17, 1703177. <https://doi.org/10.3389/fpsyg.2026.1703177>
- [6] Guss, S. S., Clements, D. H., Sharifnia, E., Sarama, J., Holland, A., Lim, C.-I., & Vinh, M. (2024). Designing inclusive computational thinking learning trajectories for the youngest learners. *Education Sciences*, 14(7), 733. <https://doi.org/10.3390/educsci14070733>
- [7] Jean, D., Stein, G., Broll, B., & Lédeczi, Á. (2025). Easing the block-to-text transition: A scaffolded approach to learning Python. In *Proceedings of the ACM Global Computing Education Conference 2025—Volume 1* (pp. 113–119). <https://doi.org/10.1145/3736181.3747126>
- [8] Kutay, E., & Oner, D. (2022). Coding with Minecraft: The development of middle school students' computational thinking. *ACM Transactions on Computing Education*, 22(2), Article 21. <https://doi.org/10.1145/3471573>
- [9] Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 33(1), 159–174. <https://doi.org/10.2307/2529310>
- [10] Liao, J., Zhong, L., Zhe, L., Xu, H., Liu, M., & Xie, T. (2024). Scaffolding computational thinking with ChatGPT. *IEEE Transactions on Learning Technologies*, 17, 1628–1642. <https://doi.org/10.1109/TLT.2024.3392896>
- [11] Long, D., & Magerko, B. (2020). What is AI literacy? Competencies and design considerations. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (pp. 1–16). Association for Computing Machinery. <https://doi.org/10.1145/3313831.3376727>
- [12] Lynn, M. R. (1986). Determination and quantification of content validity. *Nursing Research*, 35(6), 382–386. <https://doi.org/10.1097/00006199-198611000-00017>
- [13] Mladenović, M., Žanko, Ž., & Zaharija, G. (2024). From blocks to text: Bridging programming misconceptions. *Journal of Educational Computing Research*, 62(5), 1082–1106. <https://doi.org/10.1177/07356331241240047>

- [14] Morse, J. M. (1991). Approaches to qualitative-quantitative methodological triangulation. *Nursing Research*, 40(2), 120–123. <https://doi.org/10.1097/00006199-199103000-00014>
- [15] Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.
- [16] Polit, D. F., Beck, C. T., & Owen, S. V. (2007). Is the CVI an acceptable indicator of content validity? Appraisal and recommendations. *Research in Nursing & Health*, 30(4), 459–467. <https://doi.org/10.1002/nur.20199>
- [17] Romero, M. (2025). From consumption to co-creation: A systematic review of six levels of AI-enhanced creative engagement in education. *Multimodal Technologies and Interaction*, 9(10), 110. <https://doi.org/10.3390/mti9100110>
- [18] Shneiderman, B. (2020). Human-centered artificial intelligence: Reliable, safe & trustworthy. *International Journal of Human-Computer Interaction*, 36(6), 495–504. <https://doi.org/10.1080/10447318.2020.1741118>
- [19] Sun, C., Yang, S., & Becker, B. (2024). Debugging in computational thinking: A meta-analysis on the effects of interventions on debugging skills. *Journal of Educational Computing Research*, 62(4), 867–901. <https://doi.org/10.1177/07356331241227793>
- [20] Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <https://doi.org/10.1145/1118178.1118215>
- [21] Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, 17(2), 89–100. <https://doi.org/10.1111/j.1469-7610.1976.tb00381.x>
- [22] Yusoff, M. S. B. (2019). ABC of content validation and content validity index calculation. *Education in Medicine Journal*, 11(2), 49–54. <https://doi.org/10.21315/eimj2019.11.2.6>
- [23] Zhong, T., Zhu, G., Lim, K. Y., & Ong, Y. S. (2024). *Generative AI as a tool or leader? Exploring AI-augmented thinking in student programming tasks*. arXiv preprint. <https://doi.org/10.48550/arXiv.2411.19490>

Acknowledgment

The author sincerely thanks the participating students, their parents or guardians, the teachers, and the administration of the participating primary school for their cooperation and support. The author also appreciates the seven experts who contributed their time and professional expertise to the validation and refinement of the HAI-MinePy Model.

Funding Information

The author declares that no funding was received for the research, authorship, or publication of this article.

AI and LLM Disclosure (Limited Use)

Limited use of generative AI (OpenAI's ChatGPT) supported language clarity and formatting. All AI-assisted outputs were critically reviewed and validated by the author, who assumes full responsibility for the content.

Authors' Bio

Goh Kok Ming is a primary school educator based in Taiping, Perak, Malaysia. He is actively involved in educational research and innovation, with interests in artificial intelligence in education, digital competence, educational leadership, and technology-enhanced teaching and learning. His work focuses on the integration of emerging technologies such as AI, robotics, and game-based learning to support teacher professional development and improve student learning outcomes. He has presented his work at national and international conferences and is engaged in research related to digital leadership, teacher competence, and future-ready education.