Research Statement

Majeed Kazemitabaar | PhD Candidate in CS | University of Toronto

Programming languages serve as powerful user interfaces that drive automation, foster creativity, and facilitate problem-solving. However, learning to program requires a range of cognitive skills, including abstract thinking, task decomposition, and proficiency with syntax and semantics. These demands make programming challenging to learn and use effectively, particularly for novices. My research in Human-Computer Interaction (HCI) addresses these challenges by designing, developing, and evaluating innovative systems that not only make programming more accessible and engaging for beginners but also promote deeper learning and the development of computational thinking skills.

The central focus of my work addresses fundamental challenges surrounding **interaction** and **cognition** within the evolving landscape of programming with generative AI. Particularly, I have **(a)** studied the implications of AI on over-reliance when learning to code [1, 2], **(b)** developed novel interfaces and interventions that cognitively engage programmers in AI-generated solutions [4], **(c)** improved user control and verification by involving programmers in AI's step-by-step process [5], and **(d)** developed pedagogical AI assistants to support computing students in educational contexts [3]. I have also explored two additional areas of research aimed at improving the learnability of programming tools [8, 9], and fostering early computational thinking through tangible, creative programming experiences [6, 7].

Although my past work is largely centered around HCI, it intersects with disciplines like artificial intelligence, educational technology, and software engineering. My research methodology integrates established theories from learning sciences and human cognition with formative studies and participatory design sessions to inform the design and development of novel technological solutions and educational interventions. A distinct aspect of my research is conducting extensive, carefully designed experiments to evaluate the effectiveness of my solutions. I also frequently conduct field deployments in formal classrooms or museum exhibits with large groups of users to gather comprehensive qualitative insights into the implications of my solutions.

In this document, I characterize my work in two key areas: (1) designing AI tools that promote deep cognitive engagement to reduce over-reliance, support learning, and provide greater control, and (2) developing novel systems using traditional, non-AI methods to lower the barriers to programming. Looking ahead, my future research agenda will focus on advancing the development of future human-centered AI programming tools that augment rather than replace human cognition.

1 PROGRAMMING WITH AI

With the recent advancements in Artificial Intelligence, Large Language Models (LLMs) trained on code, can now generate code from natural language specifications. This shift brings programming closer to the way humans naturally think, reducing the emphasis on machine-specific algorithms and instructions. While this has the potential to make programming easier, less frustrating, and boost productivity, it also poses several cognitive challenges. Programmers may struggle to understand and verify AI-generated code, over-reliance on AI could lead to skill degradation, and the inherent vagueness of natural language specifications may result in reduced control over the AI.

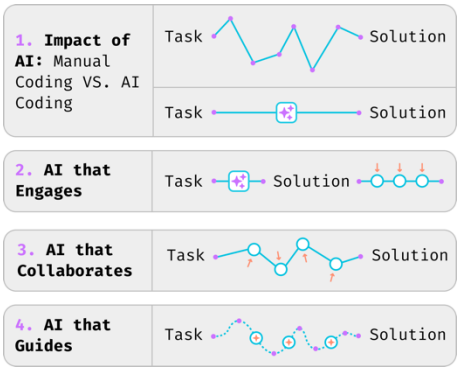

Figure 1 - Four aspects of AI-assisted programming explored in my research: **(1)** comparing two groups of novice programmers over 10 sessions, one with access to AI and one without, to understand the impact of AI on over-reliance and learning outcomes; **(2)** designing interventions that enhance metacognitive engagement by encouraging novice programmers to critically evaluate AI-generated solutions; **(3)** developing novel interaction paradigms where, instead of solving coding tasks autonomously, the AI involves users in its step-by-step process, enabling them to verify and edit its assumptions; and **(4)** creating and deploying pedagogical coding assistants that promote independent problem solving by guiding students rather than directly displaying code solutions.

The core part of my dissertation research tackles these fundamental challenges. Through a series of experiments and design interventions, I investigated four critical aspects (see Figure 1). First, I studied the impact of access to AI on over-reliance and programming skill development. Second, I developed interventions to cognitively engage users, enabling them to critically evaluate AI-generated solutions. Third, I developed novel interaction paradigms that provide users greater control by incorporating them into the AI's step-by-step problem-solving process. Finally, I created tools that promote independent problem solving in educational settings by guiding students rather than directly displaying AI-generated solutions. My research in this area is among the most influential and highly cited research papers in HCI + AI. I discuss these contributions and their implications below.

1.1 Investigating the Impact of AI on Learning Outcomes

The rapid integration of AI into programming environments through tools like Github Copilot has raised an important question: What is AI's impact on learning outcomes, particularly when AI assistance is removed later? Can AI effectively

support novices, or does it lead to over-reliance and skill degradation? I addressed this fundamental question in early 2022, which was the first study of its kind. I developed a self-paced learning platform with an embedded AI code generator powered by GPT-3, that generated code from natural language descriptions. In a three-week experiment with 69 novice learners, half had access to the AI code generator when learning to code while the other half did not. To evaluate learning outcomes, their coding performance without AI was evaluated in the last two sessions–one conducted a day later and another evaluation a week later.

The results were promising: AI assistance significantly improved learning outcomes for learners with a stronger conceptual background in programming and reduced overall frustration [1]. This work led to becoming the second most highly cited publication (**#2** of 879 accepted papers) at ACM CHI 2023 (the premiere venue for HCI research), gaining recognition across disciplines such as HCI, computing education, software engineering, educational technologies, and artificial intelligence.

Building on these findings, I conducted a follow-up analysis to better understand how participants used AI to solve coding tasks, how they interacted with it, and how their coding approaches impacted their ability to subsequently modify or extend AI-generated code. Specifically, I wanted to identify scenarios in which learners over-relied on AI and those in which novices managed to self-regulate effectively. A key observation was that many novices copied or wrote the entire task description for the AI to solve the entire task for them, which led to the poorest performance during manual code modification tasks [2]. This insight led to two further research projects, which I will describe in the next section, focusing specifically on addressing the challenges where the AI autonomously solves entire problems for users.

1.2 Promoting Cognitive Engagement when Using AI

When AI solves coding tasks autonomously without user involvements, two main risks arise: difficulty in controlling and verifying output, and an illusion of learning where users accept AI-generated code without truly understanding it.

To identify the pain points in verification and control, I conducted a formative study that revealed key challenges faced by programmers. These included the need to infer the AI's underlying assumptions and reasoning for the generated code, feeling overwhelmed by lengthy responses, and lacking fine-grained control over the AI's process. To address these issues, I proposed a novel interaction paradigm using chain-of-thought reasoning, where the AI incrementally solves tasks, actively involves the user, and displays its reasoning in an editable, structured user interface inspired by direct manipulation principles from HCI. I developed two systems embodying this interaction paradigm (Figure 2), and a controlled experiment demonstrated that it significantly enhanced users' ability to control and verify AI's output [5].

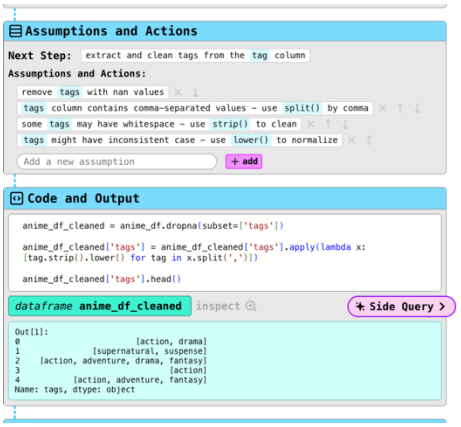

Figure 2 - A single step in the AI's problem-solving process, where it generates editable assumptions and actions for solving that step of the task. Users can inspect, verify, or edit these to regenerate the step based on updates before proceeding to the next step.

To mitigate the illusion of learning, I explored embedding carefully designed "frictions" in AI-generated code to encourage users in critically analyzing the code before they can use it. I systematically explored the design space and implemented seven cognitive engagement techniques each with distinct types of friction (Figure 3). These included typing over AI-generated code, rearranging scrambled code lines, tracing code, answering code-related questions, and prompting what should be done at each step of the code (before revealing it). In two carefully designed experiments, I found that involving the user in the step-by-step problem-solving process, where learners engage in an interactive dialog with the AI, prompting what needs to be done at each stage before the corresponding code is revealed is the most effective technique. Having the highest learning impact with the lowest amount of friction and highest correlation between perceived learning and actual learning outcomes [4].

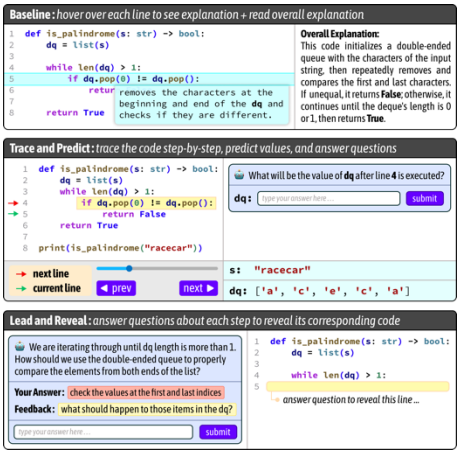

Figure 3 - The baseline shows AI-generated code without user engagement. The second intervention asks programmers to trace the code and predict variable values. The third, guides users in the step-by-step process of task, requiring them to explain each step before revealing its corresponding code.

A key implication of these projects is that involving users in the AI's step-by-step problem-solving, rather than allowing autonomous task completion, enhances verification, control, learning outcomes, and metacognitive awareness. This interactive approach fosters deeper cognitive engagement, improving user oversight, comprehension, and retention.

1.3 Designing Pedagogical AI Coding Assistants

In addition to the learning and control challenges mentioned above, the rapid rise of generative AI has also raised concerns about ethics and academic integrity in educational settings, leading many instructors to prohibit its use. However, AI has the potential to scale instructional support, lower barriers to entry, and foster independent problem-solving by guiding learners without directly revealing solutions. To explore this, I developed CodeAid (Figure 4), in early 2023 that generated pedagogical responses instead of complete code solutions. I deployed it in a large CS class with 750 students over 12 weeks and iteratively refined the tool through interviews, usage data, and feedback. Notably, our results showed that CodeAid provided a safe space for students to seek help and self-reported women, who made up only 30% of the class, used it nearly twice as much as self-reported men. This indicates the potential of personalized AI tutors to enhance equitable access to educational support and address gender-related disparities. This work, now the most highly cited CHI 2024 paper (**#1** of 1060 accepted papers), outlines key design considerations for future educational AI assistants [3]. Notably, it led to significant collaborations, including a $300,000 grant proposal to integrate AI into computing classes and the advancement of Code.org's next-generation AI Tutor.

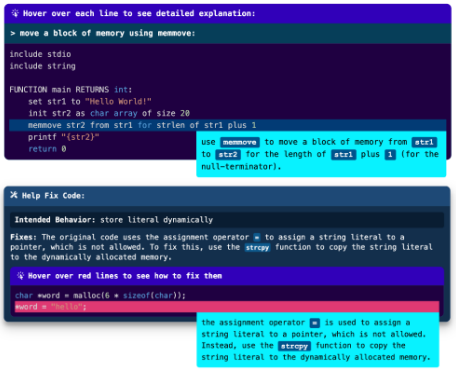

Figure 4 - CodeAid's pedagogical responses displays interactive pseudocode instead of direct code and highlights incorrect lines of student code with suggested fixes instead of directly displaying the fixed code.

2 OTHER RESEARCH AREAS

Despite extensive research in end-user programming, text-based programming environments remained largely unchanged for novice programmers. My work in this area is based on the philosophy that programming environments should include novice-to-expert scaffoldings to enable beginners to write functional code with minimal training. This led to integrating software learnability techniques like "training wheels" directly into programming interfaces. These temporary guardrails, reduce errors and enable novices to learn from mistakes as they gradually become independent. Therefore, I developed CodeStruct (Figure 5) which uses a structured editor to provide guardrails for writing code, provides immediate feedback, and visual aids. It also includes a context-aware toolbox that suggests valid code expressions that could be written at the cursor, providing in-situ learning with integrated worked examples. In a study with 26 children over 11 sessions, half transitioning from block-based to text-based programming with CodeStruct [8, 9] and half without, showed significantly less frustration without sacrificing learning outcomes. This research offered insights into the effective use and removal of short-term scaffolding. I envision future AI-powered scaffoldings to provide adaptive support for novices as they progress in the novice-to-expert transition.

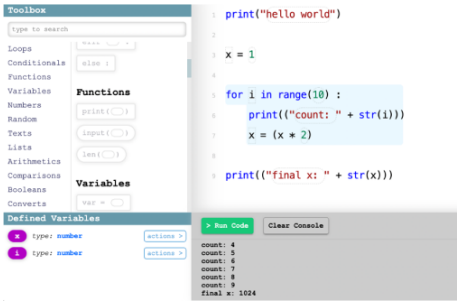

Figure 5 - CodeStruct: the context-aware toolbox enables novices to write code in a manner similar to block-based programming environments (e.g., Scratch). Additionally, the structured editor includes guardrails to minimize typing errors, visual aids to support code comprehension, and immediate feedback to fix mistakes in real-time.

In another line of research, I explored the development of wearable construction kits for young children. Prior research has showed that engaging young children in computational thinking (CT) can reduce gender-based stereotypes and provide lasting educational benefits. One effective is to introduce CT through play and creative expression. During my master's research, I undertook a two-year iterative design process to develop tangible programming tools that empower children (ages 5-10) to build their own interactive wearables. Informed by participatory design sessions with children and formative studies with STEM educators, I created MakerWear (Figure 6), which won a CHI 2017 best paper award [6]. MakerWear is a library of 32 electronic modules that enables children to integrate them into clothing and create interactive behaviors. I evaluated MakerWear in museum exhibits and workshops and showed how it facilitates the development of CT skills like sequencing, branching, problem solving, and debugging. Children also engaged in developing their own open-ended final projects in several themes including sports/fitness, role-play characters, socio-dramatic play, and decoration. Additionally, in subsequent, unpublished work, I explored hybrid visual-tangible programming approaches using a programmable module with a block-based programming interface for creating more complex behaviors.

Figure 6 - The MakerWear system consists of 32 modules: 12 sensors (black), 9 actions (white), 7 modifiers (blue), 3 miscellaneous (orange), and 1 power (red). To the right, examples of interactive designs created by children using MakerWear during our multi-session workshops are displayed.

3 FUTURE RESEARCH AGENDA

My long-term research goal is to improve how humans program computers using AI, making the process more efficient, accessible, and adaptable to diverse needs, while advancing how we learn the skills required to navigate this evolving paradigm. Achieving this vision necessitates foundational research in human-AI interaction and educational innovation—areas where an academic career provides the ideal platform to address these challenges. Below, I outline the key directions of my future research agenda.

Future of Programming with AI: An important next step is to investigate how programming practices will evolve with the integration of AI, examining implications for software engineers, end-user programmers, hobbyists, and conversational programmers. Central to this research path is identifying the skill sets essential for different programmer types and discerning which traditional skills may become obsolete. Key questions include: What are the long-term effects of widespread AI integration in programming, particularly if, within the next five years, AI becomes a universal coding assistant? How can AI-assisted programming tools be designed to meet the unique needs of diverse programmer groups? What interaction paradigms will be used by different types of programmers? For instance, tools for end-user programmers might prioritize intuitive interfaces and natural language interactions, while those for software engineers could focus on advanced debugging, algorithm selection, and system design. This research path will also explore tools and strategies to prepare and support the next generation of programmers as they navigate this technological transition.

Learning Higher-Order Skills in the Age of AI: As AI increasingly manages lower-level coding tasks, it is essential to understand how novice programmers can effectively learn higher-order skills like task decomposition, problem-solving, verification, and debugging with AI. A promising approach involves creating AI-powered programming exercises that guide users through these skills in a structured, step-by-step manner. For example, such exercises could help users break tasks into subtasks, choose appropriate data types, and evaluate the trade-offs of various algorithms and approaches. This scaffolded process ensures that learners focus on one key step at a time, with AI providing personalized feedback at each stage before moving on to the next step. To maximize the impact of these tools, research is needed to assess their effectiveness and refine them through longitudinal and comparative studies across diverse contexts. This effort will help identify best practices for fostering the mastery of higher-order programming skills when using AI.

Adaptive, Personalized AI Tutors: A key research direction lies in advancing AI-powered tutoring systems that provide personalized, adaptive learning experiences. This involves several critical components: (1) developing systems that maximize learning outcomes by tailoring instruction to individual users, dynamically assessing skill levels, and delivering targeted, context-specific feedback; (2) addressing emotional and motivational factors by incorporating personalized gamification elements and exploring the impact of proactive versus reactive AI engagement on user motivation; and (3) fostering inclusivity by designing AI tutors that create safe, supportive environments, with a focus on accessibility and empowering underrepresented groups. These advancements aim to make learning with AI both effective and equitable for diverse users.

AI-Powered Creativity for Children: Lastly, inspired by Seymour Papert's theory of constructionism, this research path aims to equip K-12 students with tools to create AI-powered applications, fostering creativity and AI literacy. The objective is to develop educational platforms similar to Scratch, that provide user-friendly interfaces for prompt engineering various modalities of generative AI models. These tools would enable children to design their own AI-driven simulations, games, storytelling projects, citizen science initiatives, and interactive systems, bridging traditional programming with AI paradigms. Emphasis will be placed on integrating these tools into school curricula to reach diverse audiences, with research evaluating their effectiveness in enhancing learning outcomes, engagement, and technical fluency among young learners.

References

[1] **Kazemitabaar, M.**, Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., & Grossman, T. (2023, April). *"Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming."* In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (pp. 1-23).

[2] **Kazemitabaar, M.**, Hou, X., Henley, A., Ericson, B. J., Weintrop, D., & Grossman, T. (2023, November). *"How Novices use LLM-based Code Generators to Solve CS1 Coding Tasks in a Self-Paced Learning Environment."* In Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (pp. 1-12).

[3] **Kazemitabaar, M.**, Ye, R., Wang, X., Henley, A. Z., Denny, P., Craig, M., & Grossman, T. (2024, May). *"CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs."* In Proceedings of the CHI Conference on Human Factors in Computing Systems (pp. 1-20).

[4] **Kazemitabaar, M.**, Huang, O., Suh, S., Henley, A. Z., & Grossman, T. (2024). *"Exploring the Design Space of Cognitive Engagement Techniques with AI-Generated Code for Enhanced Learning."* In Proceedings of the 30th International Conference on Intelligent User Interfaces.

[5] **Kazemitabaar, M.**, Williams, J., Drosos, I., Grossman, T., Henley, A. Z., Negreanu, C., & Sarkar, A. (2024, October). *"Improving Steering and Verification in AI-Assisted Data Analysis with Interactive Task Decomposition."* In Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (pp. 1-19).

[6] **Kazemitabaar, M.**, McPeak, J., Jiao, A., He, L., Outing, T., & Froehlich, J. E. (2017, May). *"MakerWear: A Tangible Approach to Interactive Wearable Creation for Children."* In Proceedings of the 2017 chi conference on human factors in computing systems (pp. 133-145).
Best Paper Award (top 1%, 2400)

[7] **Kazemitabaar, M.**, He, L., Wang, K., Aloimonos, C., Cheng, T., & Froehlich, J. E. (2016, May). *"ReWear: Early Explorations of a Modular Wearable Construction Kit for Young Children."* In Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (pp. 2072-2080).
Best Late-Breaking Work Award (top 1%, 4 / 647)

[8] **Kazemitabaar, M.**, Chyhir, V., Weintrop, D., & Grossman, T. (2022, June). *"CodeStruct: Design and Evaluation of an Intermediary Programming Environment for Novices to Transition from Scratch to Python"*. In Proceedings of the 21st Annual ACM Interaction Design and Children Conference (pp. 261-273).

[9] **Kazemitabaar, M.**, Chyhir, V., Weintrop, D., & Grossman, T. (2023, March). *"Scaffolding Progress: How Structured Editors Shape Novice Errors When Transitioning from Blocks to Text."* In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (pp. 556-562).