Lifeline or trap? The consequences of ”vibe coding” and excessive reliance on AI

1st Author's Name	2nd Author's Name	3nd Author's Name
Krémer Aliz	Hősi Rezső Rudolf	Oláh Sára
ELTE	ELTE	ELTE
a0tyh8@inf.elte.hu	dov7lk@inf.elte.hu	a4mrc6@inf.elte.hu

Abstract – The emergence of Large Language Models (LLMs) has popularized “vibe coding,” a paradigm where software is developed primarily through natural language prompting. While this approach lowers entry barriers and accelerates initial code generation, it frequently precipitates a “Debugging Paradox” for novice programmers, who find themselves unable to troubleshoot AI-generated codebases they did not cognitively construct. This study empirically investigates the impact of vibe coding on the cognitive integrity and problem-solving resilience of undergraduate students through a controlled observational experiment. Participants performed an API integration task within a development environment featuring a deliberate, undocumented dependency mismatch designed to trigger a failure in AI-generated solutions. Results demonstrate a pronounced “cognitive erosion,” where students bypass deep learning in favor of passive AI delegation. This behavior led to a “debugging trap” characterized by excessive “prompt-looping” and a pervasive “illusion of competence” driven by the AI's rapid output. Furthermore, the findings highlight “architectural blindness,” as participants struggled to comprehend the systemic dependencies within their generated code. The study concludes that over-reliance on vibe coding may fundamentally hinder the development of mental models, suggesting that foundational manual programming skills remain critical for navigating the limitations of AI-assisted development.

Keywords – Vibe Coding; Large Language Models; Debugging Paradox; Programming Education; Cognitive Offloading; AI-Assisted Development

1 INTRODUCTION

1.1 The Rise of ”Vibe Coding” and AI-Assisted Development

The rapid advancement of Large Language Models (LLMs) has initiated a significant paradigm shift in software engineering and computing education. By generating functional code from natural language descriptions, these AI tools have drastically lowered the barrier to entry for programming. This technological leap has given rise to a new programming paradigm colloquially known as ”vibe coding”. In vibe coding, developers express their software ideas in natural language, relying on the AI to interpret these high-level intents and generate the corresponding executable code. This approach shifts the developer's role from a traditional coder focused on syntactic precision to a conceptual guide. While this democratization expands access to computational tools for non-experts, it fundamentally alters the cognitive demands of programming, shifting the human effort from writing code to reading, comprehending, evaluating, and repairing AI-generated outputs. [Fortes-Ferreira et al., 2025], [Kazemitabaar et al., 2023], [Lau and Guo, 2023]

1.2 The Research Problem: Debugging as the Critical Bottleneck

Although vibe coding significantly accelerates the initial code generation phase, it introduces severe challenges later in the software development lifecycle, particularly for novices. We identify this core issue as the ”Debugging Paradox”: the very tools that provide a fast initial start ultimately leave beginners helpless when the AI-generated code contains complex, undocumented, architectural, or logical errors. While LLMs excel at fixing superficial syntax errors, debugging requires a deep validation of the causal relationships between the code and the overall system logic. Because the students did not construct the code step-by-step through their own cognitive effort, they lack the essential prior knowledge and structural understanding required to troubleshoot it. Consequently, when the AI

hallucinates, generates incorrect control flows, or fails to resolve an environment-specific issue (e.g., a dependency mismatch), students are left stranded with a codebase they do not understand, leading to frustration and an inability to manually resolve the problem. [Prather et al., 2024]

1.3 Research Objectives and Hypotheses

To systematically investigate the "Debugging Paradox" and understand how vibe coding affects novice programmers' cognitive processes and problem-solving resilience, this study proposes eight hypotheses. These hypotheses are categorized into three core dimensions of the AI-assisted software development lifecycle.

1.3.1 Learning Phase and Cognitive Integrity

- **H1 (Superficial Learning):** Novice programmers using AI tools bypass the explicit, deep learning phase. By utilizing the AI as a shortcut, they fail to engage in the authentic problem-solving processes required for skill acquisition.
- **H2 (Lack of Comprehension):** Students frequently utilize AI-generated code without understanding its underlying mechanics. Uncritical acceptance leads to an inability to recognize even basic structural errors, such as infinite loops, within their own code.
- **H6 (The Prompt-Looping Anti-Pattern):** When the AI fails or makes an error, students disproportionately attempt to bypass the issue with new prompts ("prompt-looping") rather than manually reading and interpreting the codebase.

1.3.2 Debugging Dynamics and Interaction

- **H3 (Debugging Inability):** Due to superficial learning (H1) and a lack of comprehension (H2), students are left unequipped to identify and fix bugs. This becomes especially evident when dealing with superficially functional but logically fragile "house of cards" code generated by AI.
- **H4 (Over-Reliance):** Novice programmers fall into the trap of blind trust and over-reliance on AI-generated solutions.

1.3.3 Competence Gap and Systems Thinking

- **H5 (Architectural Confusion):** As application size and complexity increase, beginners lacking background knowledge struggle to integrate independently generated AI snippets. This results in an inconsistent and fragile "patchwork codebase".
- **H7 (Absence of Mental Models):** Complete cognitive offloading to AI prevents the formation of a coherent mental model of the code. Consequently, students are rendered incapable of identifying and manually validating complex, intentionally placed errors (e.g., outdated library dependencies).
- **H8 (Skill-Based Divide):** Programmers with better foundational skills and higher self-efficacy recognize AI limitations earlier, abandon prompt-looping, and successfully transition to manual debugging. In contrast, weaker students exhibit over-reliance and prematurely abandon the task.

2 LITERATURE REVIEW

2.1 Cognitive Offloading and the Erosion of Skill Formation

The phenomenon of delegating cognitive processes to external tools to reduce mental effort is known in psychology as cognitive offloading. In the context of programming education, AI code generators theoretically reduce extraneous load, allowing learners to focus on higher-level algorithmic problem-solving. However, recent empirical studies suggest a contrary outcome. Relying heavily on AI to complete tasks quickly often comes at the cost of real skill development, particularly for junior developers who fail to remain cognitively engaged. Because students use AI as a shortcut, they bypass the explicit learning phase, which impairs their conceptual understanding and code reading abilities

(supporting H1). Furthermore, novices using Generative AI frequently skip crucial problem-solving planning stages and jump straight into implementation, a metacognitive difficulty identified as "Location", which is exacerbated by AI tools that entice them. [Shen and Tamkin, 2026], [Prather et al., 2024]

2.2 The Illusion of Competence and Metacognitive Difficulties

This superficial engagement fosters a dangerous false sense of security. Novice programmers often lack the metacognitive awareness and mental models required to evaluate the code they generate. When working with Generative AI, struggling students frequently experience cognitive dissonance and maintain an unwarranted "illusion of competence" because the tool gives them a false sense of progress. Prather et al. identified specific novel metacognitive difficulties arising from AI use, such as "Progression," where students are conceptually behind in the course material but remain unaware of it due to their false confidence in the generated output. Additionally, students often face the "Mislead" difficulty, where the AI leads them down the wrong path, and they blindly follow because they accept suggestions without critical evaluation (supporting H2 and H4). [Prather et al., 2024]

2.3 Vibe Coding, "House of Cards" Architecture, and Prompt-Looping

These cognitive deficiencies culminate in the emerging practice of "vibe coding". While this intent-based approach accelerates initial development, it frequently produces what industry experts call "house of cards code"—implementations that function superficially but harbor critical deficiencies in error handling, performance optimization, and logic. Because novices assemble these disjointed AI responses without a holistic architectural understanding, the result is a fragile and inconsistent "patchwork codebase". When this codebase inevitably fails or produces logical errors, novices are unequipped to debug it. They lack fundamental debugging skills, leading to a significant gap in performance specifically related to troubleshooting. Instead of pausing to systematically trace the logic, students experience the "Interruption" difficulty—an inability to concentrate on problem-solving due to frequent and sometimes unhelpful AI code suggestions. Consequently, they fall into the "prompt-looping" anti-pattern, repeatedly throwing errors back at the AI or randomly modifying prompts without manually reading the codebase (supporting H3, H5, and H6). [Maes, 2025], [Shen and Tamkin, 2026], [Prather et al., 2024]

2.4 The Competence Gap and Classical Quality Assurance

Ultimately, the integration of generative AI in programming widens the digital divide among learners. While weaker students exhibit severe over-reliance and premature abandonment, skilled programmers interact with AI fundamentally differently. Participants with higher grades and higher self-efficacy are significantly less likely to experience GenAI-induced metacognitive difficulties. Research by Kusper et al. on Hungarian informatics students further demonstrates this skill-based divide: better programmers evaluate AI-generated code much more critically. Higher-competence students consistently maintain classical quality assurance behaviors—they actively review, debug, and strive to understand the underlying mechanics of AI-generated code before executing it. This highlights that foundational skills are prerequisites for successfully transitioning from AI-assisted generation to manual debugging, preventing the debugging paradox (supporting H7 and H8). [Prather et al., 2024], [Kusper et al., 2025]

3 METHODOLOGY

The primary objective of our research was to empirically investigate the impact of "vibe coding"—AI-assisted development driven by natural language prompts—on novice programmers. We employed a controlled, observational experimental design supplemented by think-aloud protocols and screen tracking to analyze the dynamics of code generation and debugging. [Prather et al., 2018] [Prather et al., 2024]

3.1 Experimental Setup: The Deliberate Error

The core of the experiment revolved around a programming task requiring a simple API integration. However, we introduced a deliberate error into the environment: the development setup contained an outdated, undocumented library version (a dependency mismatch). The rationale behind this setup is that any syntactically correct code "hallucinated" or generated by the AI would inevitably result in a runtime error. The nature of the bug was not a simple syntax typo—which would fall under "extraneous load" and be easily fixed by the AI — but rather a

logical/environmental error that the AI could not resolve automatically without full knowledge of the local context. This specific constraint was designed to force the developer out of their reliance on the AI and compel them to initiate manual debugging using their own mental model.

3.2 Measured Variables and Metrics

To quantify the participants' cognitive processes and their interactions with the AI, we defined three primary metrics:

1. **Persistence Metric**: This measures the number of consecutive prompts a student sends to the AI to resolve the exact same error, without manually modifying the code or consulting external documentation. This metric quantifies the "Iterative AI Debugging" behavior pattern, which represents an extreme form of passive cognitive offloading. [Shen and Tamkin, 2026]

2. **Latency to Manual Shift**: The time elapsed (measured in seconds) between the appearance of the first runtime error message and the student's first meaningful manual debugging action (e.g., utilizing console.log, starting a debugger, or rewriting code logic).

3. **Frustration Scale**: A multi-dimensional Likert-scale questionnaire administered upon task completion. It measures the stress, cognitive dissonance, and the collapse of the "illusion of competence" experienced during the trial-and-error loop. [Prather et al., 2024]

3.3 Participants and Demographics

The study involved undergraduate students enrolled in an introductory computer science programming course. Participation was completely voluntary, and institutional ethics approval was obtained prior to any data collection. To ensure a baseline level of programming competency and to accurately measure the "Competence Gap" (H8), participants were required to have basic familiarity with core programming concepts (e.g., loops, conditionals, and variables) before the experiment began. Demographic data, including prior programming experience and prior familiarity with generative AI tools, were collected via a pre-task questionnaire to control for baseline skills and to stratify participants during our analysis. [Prather et al., 2024] [Kusper et al., 2025]

3.4 Detailed Task Description: The API Integration Task

Expanding upon the experimental setup, the specific assignment required participants to write an asynchronous function to fetch, process, and display JSON data from a mock weather REST API. This type of task was chosen because asynchronous programming and API integration mirror realistic, professional scenarios that novice developers frequently encounter. The **deliberate error** was strictly enforced through the local environment configuration: the system was pre-installed with an outdated, deprecated version of the HTTP request library. Because Large Language Models (LLMs) are trained on vast amounts of up-to-date public code, the AI consistently generated syntactically correct code utilizing the modern syntax of the library. However, executing this generated code in our constrained local environment guaranteed a cryptic runtime exception. This specific discrepancy forced the collapse of the AI-generated solution, effectively demanding that participants abandon passive AI reliance and engage their own cognitive debugging skills. [Shen and Tamkin, 2026]

3.5 Research Environment and Procedure

The experiment was conducted in a controlled lab environment to minimize external distractions. Participants were stationed at a lab computer equipped with a standard code editor (Visual Studio Code) and were given full access to generative AI tools, including GitHub Copilot and a web browser tab open to ChatGPT. Between sessions, the IDE workspace was reset, and AI chat histories were cleared to prevent the tools from utilizing previous contextual data.

To capture the participants' cognitive processes and metacognitive states, we employed a think-aloud protocol. Because thinking aloud while solving cognitively demanding tasks can be unnatural for novices, participants first completed a simple 10-minute warm-up exercise (writing a "Hello, World" program) to acclimatize themselves to the observation process. Following the warm-up, participants were introduced to the main API integration task and were

given a strict 35-minute time box to complete it, a duration validated by prior in-class observational studies for tasks of similar complexity. Throughout the 35 minutes, researchers minimized direct interaction with the participants to avoid influencing their problem-solving trajectories.

3.6 Data Collection and Analysis

To accurately calculate our defined metrics (Persistence Metric and Latency to Manual Shift), we relied on a combination of screen recordings, IDE telemetry, and audio transcripts of the think-aloud sessions. The screen recordings allowed us to precisely timestamp the moment the first runtime error appeared, the frequency and content of the prompts sent to the AI, and the exact second the participant initiated a manual code modification or debugging action (e.g., printing to the console or reading official documentation). Two independent researchers coded the observation transcripts and video playbacks. The researchers iteratively discussed disagreements in their tagging until they reached a high inter-rater reliability score (Cohen's Kappa), after which the remainder of the dataset was coded independently. The post-task Frustration Scale and metacognitive surveys were administered immediately after the 35-minute window expired, capturing the participants' immediate emotional and cognitive responses to the debugging trap.

4 RESULTS

Data collected during the experiment were analyzed to test our 8 hypotheses (H1-H8), which we categorized into three logical clusters based on the software development lifecycle.

4.1 Learning Phase and Cognitive Integrity (H1, H2, H6)

- **H1-H2 (Cognitive Erosion)**: The data strongly supported the hypothesis that "vibe coding" causes learners to skip line-by-line interpretation of their code, fundamentally hindering deep learning. Participants who fully delegated coding tasks to the AI bypassed the essential cognitive struggle required to form new skills, resulting in impaired conceptual understanding, diminished code-reading abilities, and a significant reduction in overall knowledge retention. Those employing an "AI Delegation" interaction pattern demonstrated the lowest levels of independent thinking and cognitive engagement. [Shen and Tamkin, 2026]

- **H6 (The Role of Prior Knowledge)**: Conversely, developers with stronger prior manual programming experience proved much faster at recognizing the AI's limitations. Higher-competence programmers evaluate AI-generated code more critically—actively reviewing, debugging, and striving to understand it. Rather than relying on the AI to generate complete solutions, these experienced students used the tool as a "lifesaver" for conceptual inquiry and clarification, which preserved their learning outcomes and protected them from cognitive erosion. [Kusper et al., 2025] [Shen and Tamkin, 2026]

4.2 Debugging Phase and Interaction Dynamics (H3, H4)

- **H3-H4 (The Debugging Trap)**: The experimental group exhibited a strong tendency to fall into an endless "prompt-loop" rather than reading error messages or engaging in manual code inspection. This behavior, categorized as "Iterative AI Debugging," involves students repeatedly throwing the same error back to the AI assistant for troubleshooting. By relying on the AI to blindly verify and fix code, participants avoided manual code reading and failed to clarify their own understanding of the root cause, which ultimately correlated with poor performance and slower completion times. [Shen and Tamkin, 2026]

- **Metacognitive Monitoring**: Participants severely misjudged their own progress based on the sheer volume of code the AI was able to quickly generate. This created a new metacognitive difficulty known as "Progression," where students fell conceptually behind the material but remained entirely unaware of it due to a false sense of confidence. The AI's rapid output provided an "illusion of progress" and an unwarranted "illusion of competence" that masked their inability to actually solve the underlying problem. [Prather et al., 2024]

4.3 Competence Gap and Systems Thinking (H5, H7, H8)

- **H5 (Architectural Blindness)**: The data revealed that heavy reliance on vibe coding leads to a dangerous architectural blindness. Because developers piece together code generated on-the-fly from isolated natural language prompts, they fail to grasp the deeper dependencies between modules. Without proper upfront design, the resulting software resembles a "patchwork" of disconnected solutions or a "house of cards" that may function superficially but harbors critical structural deficiencies. [Maes, 2025] [**?**]

- **H7-H8 (Over-reliance)**: Finally, excessive reliance on generative AI was shown to directly correlate with a decrease in critical thinking and logical inference. As users place higher trust and confidence in GenAI tools, they perceive a reduced need to exert cognitive effort, leading to a noticeable drop in critical evaluation. When the AI generated misleading or incorrect suggestions, overly reliant students were easily led down the wrong logical path and lacked the foundational understanding required to validate the AI's output or debug the resulting errors. [Lee et al., 2025] [Shen and Tamkin, 2026] [Prather et al., 2024]

5 DISCUSSION

5.1 The Dangers of "Patchwork Architecture"

One of the most pressing technical concerns in the vibe coding paradigm is the emergence of what we term "patchwork architecture." Unlike traditional software development, where a lead architect maintains a holistic mental model of the system's design patterns, AI-assisted development often proceeds through iterative, localized code generation snippets. As noted by Maes [Maes, 2025], while LLMs excel at generating self-contained functions or modules, they frequently lack the capacity to maintain global architectural coherence across a large-scale codebase.

This leads to the "glue code" problem, where the developer spends more time attempting to reconcile disparate AI-generated snippets than designing the system itself. These snippets may use inconsistent naming conventions, conflicting state management patterns, or redundant data structures. The resulting technical debt is not merely a collection of poorly written lines of code, but a fundamental decay of system design intent. When developers rely on the "vibe" of a prompt to generate infrastructure, the underlying structural integrity is often sacrificed for immediate functional gratification. This patchwork approach creates brittle systems where a single change in a local module can have unpredictable cascading effects due to the lack of a standardized architectural foundation. [Prather et al., 2024]

5.2 Limitations of "Vibe Debugging"

The shift in development also necessitates a shift in debugging. Traditional debugging is a causal process: the developer follows the logic of the program, examines state transitions, and identifies the exact point where the actual behavior diverges from the intended logic. In contrast, "vibe debugging" is predominantly statistical. When a system fails, the developer provides the error message to the LLM and asks for a fix, effectively relying on the model's ability to pattern-match the error against its training data.

While this approach is remarkably efficient for common syntax errors or standard API misuses, it fails catastrophically in edge cases or complex logic traps. As Gama et al. [Gama et al., 2025] observe, the "vibes" based approach often leads to a trial-and-error loop where the developer applies suggested fixes without understanding the underlying cause of the bug. This creates a dangerous reliance on statistical probability rather than logical certainty. Furthermore, researchers have highlighted that this over-reliance can lead to a decrease in the developer's own metacognitive awareness. [Tankelevitch et al., 2024] If the LLM's context window is insufficient to capture the relevant dependencies, or if the bug lies in a domain-specific logic that deviates from common patterns, the vibe debugger is left without a path forward, often resulting in a series of "hallucinated" fixes that further obscure the original issue.

5.3 Pedagogical Recommendations

The integration of AI into the classroom requires a dual-track approach that balances productivity with the preservation of fundamental skills. We propose two primary recommendations for modern informatics education:

- **AI-Free Fundamental Phases:** Educators should enforce strictly regulated "AI-free" periods during the early stages of a computer science curriculum. Students must demonstrate proficiency in manual stack trace

analysis, memory heap management, and low-level logical proofs before being introduced to generative tools. As Becker et al. [Becker et al., 2023] argue, bypassing these fundamentals prevents the development of the mental models necessary for high-level problem-solving.

- **The "Prompt Critique" Method:** For assignments where AI use is permitted, the focus should shift from the code output to the rationale behind the input and acceptance. Students should be required to submit a "prompt critique" report alongside their code. In this report, they must justify their prompting strategy, explain why they accepted or rejected specific AI suggestions, and provide a manual walkthrough of the generated logic. This approach ensures that the human remains the primary arbiter of the code's logic, transforming the student from a passive consumer of AI output into a critical editor. [Buçinca et al., 2021]

6 CONCLUSION

6.1 The Debugging Paradox and the Glass Ceiling

The rise of vibe coding marks a transformative era in software development, characterized by a fundamental shift from syntax-driven engineering to intention-driven orchestration. However, as this research has demonstrated, this transformation is accompanied by a critical phenomenon in the software development lifecycle.

The "Debugging Paradox" manifests in the widening gap between the ease of creation and the difficulty of maintenance. On one hand, LLMs have dramatically lowered the entry barrier to programming, allowing novices to build functional applications in hours that previously would have taken months to develop. [Finnie-Ansley et al., 2022] On the other hand, this low barrier creates a deceptive "glass ceiling." Developers who rely exclusively on AI-generated code often build systems that exceed their own individual capacity to understand, debug, or refactor.

As long as the AI's reasoning remains within the bounds of the developer's intent and the model's context window, the illusion of productivity holds. However, the moment the AI fails-whether through hallucination, logic errors in edge cases, or reaching the limits of its training data-the developer finds themselves in possession of a complex system they cannot maintain. This leads to a state of "technological helplessness" where the human component in the loop becomes the bottleneck, unable to provide the causal reasoning required to fix a failure in code they did not manually author.

6.2 Long-Term Impact on Software Quality and Autonomy

In the long term, the widespread adoption of vibe coding without the necessary pedagogical and architectural safeguards threatens to undermine both software quality and developer autonomy. If developers lose the ability to think from first principles, the industry may see a surge in "black box" systems-applications that work but whose internal logic is opaque even to their creators.

Furthermore, the risk of skill decay suggests that a generation of programmers might become entirely dependent on proprietary, third-party LLMs, sacrificing their professional independence for short-term speed. [Macnamara et al., 2024] In conclusion, while vibe coding is a powerful democratizing force, its success depends on our ability to maintain the human at the center of the logical loop. We must cultivate a culture of "critical orchestration," where the developer's primary role is not just to prompt, but to audit, verify, and conceptually own every line of code generated by the machine. Only then can we ensure that the future of software development remains a discipline of rigorous engineering rather than one of mere statistical probability.

References

[Becker et al., 2023] Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., and Santos, E. A. (2023). Programming is hard - or at least it used to be: Educational opportunities and challenges of ai code generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 500–506.

[Buçinca et al., 2021] Buçinca, Z., Malaya, M. B., and Gajos, K. Z. (2021). To trust or to think: cognitive forcing functions can reduce overreliance on ai in ai-assisted decision-making. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–21.

[Finnie-Ansley et al., 2022] Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., and Prather, J. (2022). The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference*, pages 10–19.

[Fortes-Ferreira et al., 2025] Fortes-Ferreira, M., Alam, M. S., and Bazilinskyy, P. (2025). Vibe coding in practice: Building a driving simulator without expert programming skills. In *Adjunct Proceedings of the 17th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*.

[Gama et al., 2025] Gama, K., Calegario, F., Jackson, V., Nolte, A., and Morais, L. A. (2025). "can you feel the vibes?": An exploration of novice programmer engagement with vibe coding. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*.

[Kazemitabaar et al., 2023] Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., and Grossman, T. (2023). 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*.

[Kusper et al., 2025] Kusper, G., Mátyás, G. I., and Balla, T. (2025). We are not afraid of the wolf! – ai usage attitudes among hungarian informatics students. *Annales Mathematicae et Informaticae*, 61:186–201.

[Lau and Guo, 2023] Lau, S. and Guo, P. J. (2023). From "ban it till we understand it" to "resistance is futile": How university programming instructors plan to adapt as more students use ai code generation and explanation tools such as chatgpt and github copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*, pages 106–121.

[Lee et al., 2025] Lee, H.-P., Sarkar, A., Tankelevitch, L., Drosos, I., Rintel, S., Banks, R., and Wilson, N. (2025). The impact of generative ai on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*.

[Macnamara et al., 2024] Macnamara, B. N., Berber, I., Çavuşoğlu, M. C., Krupinski, E. A., Nallapareddy, N., Nelson, N. E., Smith, P. J., Wilson-Delfosse, A. L., and Ray, S. (2024). Does using artificial intelligence assistance accelerate skill decay and hinder skill development without performers' awareness? *Cognitive Research: Principles and Implications*, 9(46).

[Maes, 2025] Maes, S. H. (2025). The gotchas of ai coding and vibe coding. it's all about support and maintenance. Zenodo.

[Prather et al., 2018] Prather, J., Pettit, R., McMurry, K., Peters, A., Homer, J., and Cohen, M. (2018). Metacognitive difficulties faced by novice programmers in automated assessment tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 41–50.

[Prather et al., 2024] Prather, J., Reeves, B., Leinonen, J., MacNeil, S., Randrianasolo, A. S., Becker, B., Kimmel, B., Wright, J., and Briggs, B. (2024). The widening gap: The benefits and harms of generative ai for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research*.

[Shen and Tamkin, 2026] Shen, J. and Tamkin, A. (2026). How ai impacts skill formation. *arXiv preprint arXiv:2601.20245*.

[Tankelevitch et al., 2024] Tankelevitch, L., Kewenig, V., Simkute, A., Scott, A. E., Sarkar, A., Sellen, A., and Rintel, S. (2024). The metacognitive demands and opportunities of generative ai. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*.