Reading, Writing, and Delegating Code

What Cognitive Science Says About Learning to Program in the Age of AI Agents

Andrea Gazzi · August 2026

Abstract

AI coding agents can now generate, explain, and refactor code from natural-language instructions, and adoption among both novice and professional developers is accelerating. This raises a practical question for anyone learning to program: does relying on an agent to write code change how well a person actually learns to code? Drawing on neuroimaging research on code comprehension, classic findings on the generation effect and testing effect in memory research, a randomized controlled trial that directly tests AI-assisted skill formation, and recent empirical studies of AI coding assistants in educational and professional settings, this paper argues for a balanced position. AI agents measurably increase productivity for some tasks and users, but that productivity is uneven, and the strongest available evidence indicates that active production of code — not passive exposure to it — is what builds durable, transferable understanding. The two are not in conflict once the practice is deliberate.

1. Introduction

You describe what you want, the agent writes the code, and it works. That loop is fast, it's satisfying, and it's becoming the default way a growing share of developers — beginners included — interact with code. This raises an obvious follow-up question: if you never wrestle with the syntax yourself, are you actually learning to program, or just getting good at asking?

This paper reviews what cognitive science, memory research, and computing-education studies currently say about that question, and lays out a practical, non-alarmist position: agents and deliberate practice can coexist, but only if the second one is protected on purpose.

2. Your Brain Doesn't Read Code Like It Reads English

Code looks language-like, so an early hypothesis in cognitive neuroscience was that reading code recruits the same brain regions as reading a sentence. A functional MRI study led by MIT's Anna Ivanova and Evelina Fedorenko tested this directly, having programmers read and evaluate short snippets of Python and a visual language called ScratchJr while their brain activity was recorded (Ivanova et al., 2020). The result ran against that hypothesis: comprehending code activated the brain's “multiple demand” network — a general-purpose system also used for math, logic, and problem-solving — rather than the left-hemisphere language regions used for natural

language. Even so, code comprehension did not fully overlap with math and logic processing either; the authors concluded that understanding code is, in their words, its own cognitive category.

A separate neuroimaging study went further by directly comparing reading code, reading prose, and a spatial task (mental rotation) in novice programmers using functional near-infrared spectroscopy (Endres et al., 2021). The finding relevant here: brain activity during coding was more different from prose reading than it was from the spatial visualization task, with coding showing comparatively stronger engagement of regions associated with spatial reasoning and task difficulty. For beginners, writing and working with code leans on different and more effortful machinery than simply reading text — including reading code itself. This matters for the AI-agent question: if a large share of learning value comes from that effortful, spatially engaged processing, then routing around it by only reading agent-generated code may skip the part of the task doing the cognitive work.

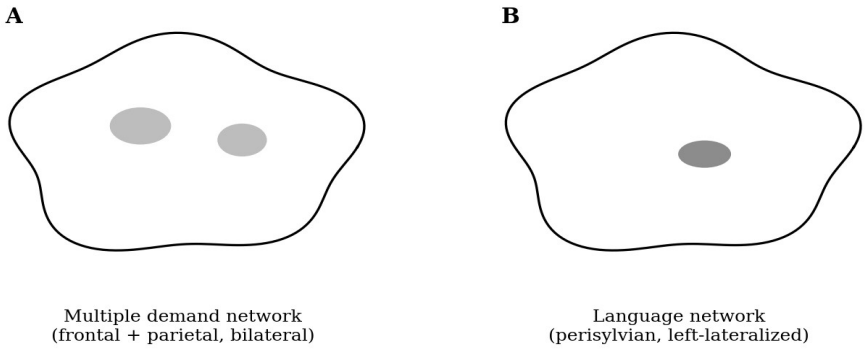

Figure 1. Code comprehension recruits the multiple demand network (panel A), the same domain-general system used for math and logic, rather than the perisylvian language network engaged during prose reading (panel B) (Ivanova et al., 2020).

3. Why Producing Beats Consuming: The Generation and Testing Effects

Independent of the coding-specific research, two well-established findings from memory science point the same direction.

3.1 The Generation Effect

People remember material they actively produce themselves substantially better than material they simply read, even when both groups spend equal time with it (Slamecka & Graf, 1978). The classic demonstration used simple word tasks — for example, generating “fast” from the cue “rapid / f__” led to far better retention than reading the completed pair “rapid – fast.” Follow-up work attributes the effect to deeper encoding: producing an answer forces more processing than recognizing one.

3.2 The Testing Effect

Sometimes called retrieval practice: actively recalling information strengthens memory more than re-reading or re-studying it, even though re-reading feels more productive in the moment (Roediger & Karpicke, 2006). In the original study, students who read a passage once and then repeatedly tried to recall it from memory outperformed students who re-read the same passage four times — the gap was small immediately afterward but large a week later, and the re-readers had wrongly predicted they would remember more. Both effects share a mechanism: the mental effort of retrieving or generating something yourself, not just being exposed to the correct answer, is what makes it stick.

3.3 A Note on What These Effects Do — and Don't — Prove

It is worth being precise about the scope of this evidence before applying it to programming. Slamecka and Graf (1978) tested memory for word pairs; Roediger and Karpicke (2006) tested memory for short factual passages. Neither study involved code, programmers, or anything resembling a software task. They establish a general mechanism — effortful retrieval and self-generation strengthen memory more than passive exposure — but extending that mechanism to "watching an AI agent write code" is an inference by analogy, not a demonstrated finding. Building an argument by connecting a 1978 word-pair study, a 2006 memory study, and 2020s neuroimaging papers produces a plausible hypothesis, not a proven causal chain. Section 4.1 introduces a study that closes this gap by testing the mechanism directly, on programmers, using a randomized design.

Applied to programming, the implication is nonetheless straightforward as a hypothesis. Watching an agent produce correct code is closer to "re-reading" than to retrieval practice: the correct answer is presented, not generated. Writing the code yourself — or, short of that, actively predicting, explaining, or reconstructing what an agent produced — is closer to the retrieval and generation conditions that these studies show produce durable learning.

Figure 2. Watching an agent produce code parallels re-reading; writing or explaining the code yourself parallels the retrieval practice shown to build durable memory (Slamecka & Graf, 1978; Roediger & Karpicke, 2006).

4. What Happens When an Agent Does the Producing

This is not just a theoretical concern; it has started showing up in direct studies of AI coding assistants. On the productivity side, the evidence is positive but uneven. A large controlled study of GitHub Copilot found developers completed a coding task 55.5% faster with the tool, with the biggest gains for less experienced programmers (Peng et al., 2023). A 2025 classroom study similarly found that computer science students completed programming tasks 34.9% faster and made 50% more solution progress when using Copilot on unfamiliar, pre-existing codebases ("brownfield" tasks), while spending measurably less time manually writing code and searching documentation themselves (Shihab et al., 2025).

These gains are not universal, however. Field studies of more experienced developers working on longer, real-world tasks have found substantially smaller effects: one study of professional developers found AI-generated code completions boosted productivity by 26.8% as measured by pull requests and commits (Cui et al., 2024), while a 2025 study of experienced open-source developers on complex, long-horizon tasks found no significant speedup — and in some cases a slowdown, attributed to time spent waiting for and reviewing AI-generated code (Becker et al., 2025). Taken together, the productivity benefit of AI coding assistance appears concentrated among novices completing short, well-scoped tasks — which matters here, because that is also the population and task type most likely to still be learning.

The 2025 Shihab et al. study is also an early empirical signal on the learning question. Alongside the performance gains, exit interviews found students expressing concern that they did not understand how or why the AI's suggestions worked — the tool had taken over not just the typing, but a share of the thinking. The researchers frame this through cognitive load theory (Sweller, 1988): Copilot clearly reduces extraneous cognitive load, the effort spent on boilerplate and syntax recall, which is a real benefit. But without deliberate structuring, it can also strip away germane cognitive load — the effortful, productive struggle that is actually where learning happens — rather than only removing the unproductive kind.

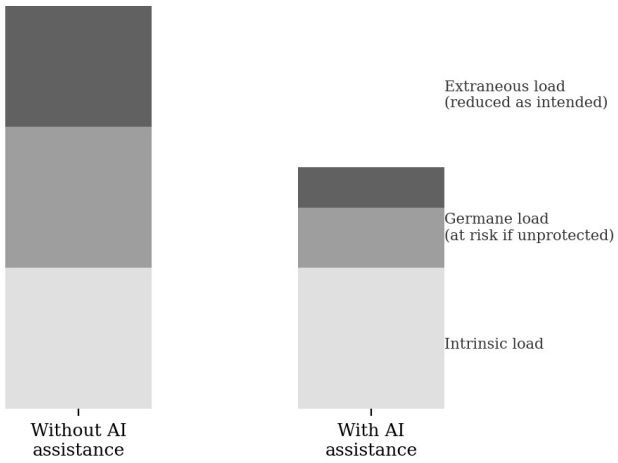

Figure 3. Cognitive load theory (Sweller, 1988) applied to AI-assisted coding. AI assistance reliably reduces extraneous load, but without deliberate practice it can also reduce germane load — the productive struggle where learning happens.

4.1 Closing the Causal Gap: A Randomized Trial on AI and Skill Formation

The studies above are informative but limited: the neuroimaging and memory research are not about coding, and the Copilot studies measure self-reported confusion or productivity, not measured skill. Shen and Tamkin (2026) close this gap directly. In a randomized controlled trial, developers with prior Python experience but no exposure to Trio, an asynchronous programming library, were assigned to complete identical coding tasks either with or without access to a chat-based AI assistant. All participants then took a quiz measuring conceptual understanding, code reading, and debugging ability.

The AI-assisted group scored 17% lower on the quiz — a statistically significant reduction — and did not complete the tasks significantly faster overall, despite the assistant being able to produce complete, correct solutions on request. Qualitative analysis of screen recordings explained why: participants who fully delegated code generation to the AI moved quickly but learned little, while participants who used the AI conversationally — asking conceptual questions or requesting explanations alongside generated code — scored nearly as well as the no-AI group, with little or no loss in speed. The no-AI group, meanwhile, encountered more real errors during the task, and the researchers tie this directly to their stronger debugging and conceptual performance: struggling with and resolving errors independently appears to be a meaningful part of how the skill was formed in the first place.

This is the first evidence discussed here that is both causal (randomized, not observational) and specific to coding skill formation rather than general memory or self-reported confusion. It

also sharpens the practical question this paper is trying to answer: the issue is not whether to use AI, but which patterns of use preserve learning.

5. A Balanced Practice, Not a Ban

None of this is an argument against using AI coding agents; the productivity data, while uneven, is strong enough for novices and short tasks that avoiding these tools outright is not a realistic recommendation. Nor does the underlying evidence suggest agents are inherently harmful — only that unexamined, purely delegated use forfeits a well-documented learning mechanism, and that this cost is now measured directly rather than only inferred. The research instead points toward specific habits that let productivity and learning coexist:

- Generate before you delegate. Attempt a solution, or at least a plan, before asking an agent — this preserves the generation effect that comes from producing an answer rather than recognizing one.
- Ask conceptual questions, not just for code. In the Shen and Tamkin (2026) trial, the participants who asked the AI to explain concepts — rather than simply generate or fix code — were the ones whose quiz scores matched the no-AI group. This is the most concrete, evidence-backed habit in this list.
- Explain the output back. After an agent produces code, restate in your own words why it works; this converts passive reading into a retrieval-practice-like act.
- Let yourself hit some errors. The no-AI group's debugging advantage in the randomized trial came directly from encountering and resolving real errors rather than having them avoided or fixed for them.
- Revisit without the agent. Periodically reproduce a previously agent-assisted solution from memory — the delayed-recall design in the classic testing-effect studies is exactly this pattern.

6. Conclusion

The evidence reviewed here varies in strength, and it is worth being clear about that rather than treating it as a single, uniform case. Understanding code draws on general-purpose problem-solving circuitry rather than language circuitry (Ivanova et al., 2020; Endres et al., 2021), and producing information yourself builds stronger memory than reading it — though that finding originates outside coding entirely (Slamecka & Graf, 1978; Roediger & Karpicke, 2006). The strongest and most directly relevant evidence is the most recent: a randomized trial showing that

fully delegating coding tasks to AI measurably reduces skill formation, while conversational, question-asking use largely does not (Shen & Tamkin, 2026). The practical conclusion is not to avoid AI agents, which is neither realistic nor supported by the productivity evidence, but to be deliberate about which parts of the coding process a learner still does themselves. Agentic development and durable skill-building are compatible; they simply are not automatic, and there is now a direct, causal basis — not just an inference from unrelated research — for which habits protect the second one.

References

Becker, J., Rush, N., Barnes, E., & Rein, D. (2025). Measuring the impact of early-2025 AI on experienced open-source developer productivity. arXiv:2507.09089.

Cui, Z. K., Demirer, M., Jaffe, S., Musolff, L., Peng, S., & Salz, T. (2024). The effects of generative AI on high skilled work: Evidence from three field experiments with software developers. SSRN 4945566.

Endres, M., Kovelman, I., Karas, Z., Hu, X., & Weimer, W. (2021). Relating reading, visualization, and coding for new programmers: A neuroimaging study. In Proceedings of the 43rd International Conference on Software Engineering (ICSE 2021). https://arxiv.org/abs/2102.12376

Ivanova, A. A., Srikant, S., Sueoka, Y., Kean, H. H., Dhamala, R., O'Reilly, U.-M., Bers, M. U., & Fedorenko, E. (2020). Comprehension of computer code relies primarily on domain-general executive brain regions. eLife, 9, e58906. https://doi.org/10.7554/eLife.58906

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. arXiv:2302.06590. https://arxiv.org/abs/2302.06590

Roediger, H. L., & Karpicke, J. D. (2006). Test-enhanced learning: Taking memory tests improves long-term retention. Psychological Science, 17(3), 249–255. https://doi.org/10.1111/j.1467-9280.2006.01693.x

Shen, J. H., & Tamkin, A. (2026). How AI impacts skill formation. arXiv:2601.20245. https://arxiv.org/abs/2601.20245

Shihab, M. I. H., Hundhausen, C., Tariq, A., Haque, S., Qiao, Y., & Mulanda, B. W. (2025). The effects of GitHub Copilot on computing students' programming effectiveness, efficiency, and processes in brownfield coding tasks. In Proceedings of the 2025 ACM Conference

on International Computing Education Research V.1 (ICER 2025). ACM. https://doi.org/10.1145/3702652.3744219

Slamecka, N. J., & Graf, P. (1978). The generation effect: Delineation of a phenomenon. Journal of Experimental Psychology: Human Learning and Memory, 4(6), 592–604.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. Cognitive Science, 12(2), 257–285.