

# AI for Me, Not (Yet) for Thee? Desirable Difficulties and Deliberate Friction with LLMs

Andrew Heiss

Andrew Young School of Policy Studies, Georgia State University  
aheiss@gsu.edu

May 1, 2026

**Abstract** I am simultaneously a critical skeptic of large language models (LLMs) and their role in the learning process, and a reluctant and wary user of those same tools in my own work. This essay is an attempt to work through this tension by examining how LLM-based tools shape and reshape the foundations of expertise required to use them. If learners can generate working code through a Google search, if they can upload a blank problem set template and get a completed assignment in seconds, and if they can design, carry out, and analyze research projects in minutes, how can they build the expertise that is required to work with LLM output? Relatedly, if experienced researchers spend the bulk of their time reviewing LLM output and not working through code themselves, what happens to their expertise? The answer to these questions, I argue, is to deliberately introduce difficulty and friction in quantitative work, allowing learners to build expertise and helping experienced researchers maintain it.

In December 2025, I was grading final projects for my course on data visualization with R. On one of my screens, I had a browser open to my university's learning management system and came across a now-familiar phenomenon: a student had used verbatim ChatGPT output for their assignment. The code they included was flawless, but used function arguments and code syntax that we never covered in class and that weren't actually necessary. More concerning, the original chat prompt was inadvertently still included, along with fill-in-the-blank sections that the large language model (LLM) had included to help tailor the response to the course (e.g. "[insert an example from your professor's class here]"). Following my course's AI policy, I gave a sizable point reduction, sighed heavily, and moved on to the next assignment.

While that next assignment loaded, I looked to my second screen where I had Claude Code—an agentic LLM-based coding tool developed by Anthropic—running in my code editor. I was using Claude to help translate a teaching demonstration that I had written in R several years ago into an interactive Javascript version that I could use in future statistics courses. The conversion took an hour of substantial back-and-forth with the LLM, but it was almost complete and the demonstration was working exactly how I had intended.

I stopped typing and stared down at my keyboard. What was I doing here? Was this just open hypocrisy? Do what I say (don't use AI), not what I do (use AI)?

I've gone through several phases in my approach to LLMs, from amusement (“haha it can write a limerick about the internal mechanics of `ggplot2`; isn't that cute!”), then to alarm and panic (“oh no, students are putting my problem set instructions into ChatGPT and having the machine do it all”), and then to curiosity (“everyone's talking about Claude Code; I guess I'll see what the fuss is all about”), and finally to wary, cautious, reluctant partial adoption.

I am a quantitative political scientist who teaches and researches with quantitative methods. I spend hours every day teaching, researching, analyzing, and writing with R, Python, and several other programming languages. I've used these computational tools for decades and have arrived at hard-won expertise through spending countless time searching Google, asking questions on online forums, poring over documentation, and working on dozens and dozens of projects. This difficulty built that expertise, and it has allowed me to evaluate LLM output and quickly recognize which parts are relevant, which parts are extraneous fluff, and which parts are wrong. Learners in an introductory class who search for an error message and then find and use LLM-generated code at the top of the search results do not have that underlying expertise and bypass the learning process.

This essay is an attempt to work through this tension by examining how LLM-based tools shape and reshape the foundations of expertise required to use them. If learners can generate working code through a Google search, if they can upload a blank problem set template and get a completed assignment in seconds, and if they can design, carry out, and analyze research projects in minutes, how can they build the expertise that is required to work with LLM output? Relatedly, if experienced researchers spend the bulk of their time reviewing LLM output and not working through code themselves, what happens to their expertise? The answer to these questions, I argue, is to deliberately introduce difficulty and friction in quantitative work, allowing learners to build expertise and helping experienced researchers maintain it.

## Forklifts in weight rooms

Like programming, education has been a central battleground in debates over the use of AI and LLMs for automation. The drive to remove friction from learning is hardly new. In the 1960s, B.F. Skinner built on work from earlier decades to develop the idea of programmed instruction, where students independently work through a sequence of scaffolded instructional chunks, responding to short check-in questions and advancing as they answer correctly (Twyman 2020). Errors are minimized by design, learners are rewarded as they progress, and mastery accumulates through correct repetition. Skinner believed that this process could scale indefinitely and that “teaching machines” could automate instruction entirely (Watters 2021). His argument has proven remarkably durable. Modern online education—including university courses, degree programs, and platforms like Khan Academy—descends from the premise of programmed instruction (Root and Rehfeldt 2021). For example, in 2024 Khan Academy partnered with Microsoft and OpenAI to create Khanmigo, a customized LLM aimed at bringing “time-saving and lesson-enhancing AI tools to millions of educators” (Beatty 2024). Khanmigo was designed to serve as a personalized tutor, walking students through difficult tasks and reducing friction from the learning process. Much of contemporary AI-based educational technology follows the same premise: keep the learner moving, keep the errors low, and keep the reward signal flowing.<sup>1</sup>

However, cognitive science has found that this type of streamlined programmed instruction tends not to create durable learning and that deeper mastery of a topic requires more than reward-based operant conditioning. Cognitive pedagogical research distinguishes between two types of learning: (1) retrieval strength, or how easily something comes to mind right now, and (2) storage strength, or how durably and flexibly knowledge is encoded for later use (Bjork and Bjork 2011). Soderstrom and Bjork (2015) argue that the goal of instruction should be long-term *learning*, or permanent changes in understanding and skills that require storage strength. However, since it’s a longer-term outcome, evidence of learning and storage strength is difficult to immediately measure. We can instead observe *performance*, or what learners immediately retrieve when tested. Consider the stereotypical example of students cramming before a midterm exam—late night flashcard memorization can boost retrieval strength by minimizing errors on a test and resulting in higher grades, but it does not typically improve storage strength or produce long-term learning.

Counterintuitively, “the conditions that produce the most errors during acquisition are often the very conditions that produce the most learning” (Soderstrom

---

<sup>1</sup>Some educational companies take this imperative to the extreme—Duolingo’s short exercises are engineered to minimize errors and maximize engagement, and its owl mascot Duo is notorious for its vaguely menacing threats to remind learners to return to the app daily (Stewart 2025).

and Bjork 2015, 176). Rather than speeding through programmed instruction by memorizing Skinner-like tasks, Bjork and Bjork (2011) find that learners achieve more permanent learning when confronted with “desirable difficulties,” or “conditions that create challenges and slow the rate of apparent learning” (Bjork and Bjork 2011, 57). These difficulties include a range of strategies that are designed to deepen storage strength, such as spaced practice (where learners have time to forget concepts) or interleaving across topics (where learners cannot reference preceding examples). This in turn forces learners to engage in the processes of retrieval and generation, which improve storage strength by modifying learners’ memories and making them more accessible in new contexts (Bjork and Bjork 2011, 61; Bjork 1975).

LLMs, however, provide on-demand retrieval and completely bypass generation, notably using their own form of generation—LLMs are also called “generative AI.” From a learner’s perspective, turning to an LLM feels like learning, but in practice, LLMs circumvent and replace *performance*, and not *learning*. In an introductory data analysis class, a learner can generate code that runs perfectly and creates a publication-worthy plot—achieving fluent, immediate high performance with no lasting, durable learning. I can now paste any of my course problem sets into an LLM chat interface and create all the code and interpretation in under a minute.<sup>2</sup> Feeding empty problem sets to an always-on and often-unreliable Answer Machine does not constitute generation. Learners do not retrieve past knowledge and recontextualize it and reshape their memories and understanding. Instead, generation is outsourced entirely to the machine.

Newer research in psychology and computer science is finding evidence for the need for retrieval and generation, particularly for beginner learners. In a study with nearly 1,000 high school students, Bastani et al. (2025) randomly assigned learners to two different versions of an LLM service: one that provided unfettered access to LLM prompts and one that was trained to act as a tutor and not give any direct answers. Both groups of students saw significant improvements in performance during LLM-assisted practice, but when access to the LLM was removed, students assigned to unlimited access condition underperformed compared to those in the tutoring-focused condition. The tutoring safeguards helped mitigate that reduction in performance, and in the absence of guardrails, learners tended to use LLMs as a “crutch” for learning. Bastani et al. (2025) conclude that though LLMs simplify many tasks and provide apparent boosts in performance, “they come with the risk of deteriorating our ability to effectively learn some of the skills required to solve these tasks” (2025, 6). In similar research sponsored by

---

<sup>2</sup>See <https://claude.ai/share/a05db93c-0008-469f-8646-f01bd77cd7ee> for an example from my casual inference course. I uploaded a Quarto file for a problem set and Claude got 90% of the assignment correct in its first try. I fixed it with one additional prompt.

Anthropic, Shen and Tamkin (2026) conducted a series of experiments to see how software developers' use of LLMs influenced their mastery of an unfamiliar Python library. Developers assigned to use AI assistance to complete new programming tasks scored significantly lower in post-task assessments, achieved no gains in productivity or efficiency, and saw an “erosion of conceptual understanding, code reading, and debugging skills” (Shen and Tamkin 2026, 18). Once again, retrieval and generation are necessary for lasting learning.

In two experiments with students in graduate-level programming courses, Lehmann et al. (2025) found that while LLM use did not affect learning outcomes in general, students who asked LLMs for solutions saw a sizable negative effect on long-term learning. Learners with less initial knowledge tended to prefer quick, immediate answers rather than exerting more focused effort for learning, and used LLMs as a substitute for learning activities as a result (Lehmann et al. 2025, 25). The repeated substitution of LLM output for actual learning activities can create a state of “cognitive surrender,” where people “readily incorporate AI-generated outputs into their decision-making processes, often with minimal friction or skepticism,” (Shaw and Nave 2026, 48) bypassing learning—and critical thought—entirely.

Learning is hard and takes time. Learning requires struggle and repetition and more struggle and more repetition. This isn't a form of programming hazing (i.e. “I had to walk to school uphill both ways in the snow and now you must too”). It's the actual process of learning and growing and developing and improving. Using LLMs to remove that struggle—especially in the absence of a strong foundation—is pedagogically costly and reduces learning. In the words of Bjork and Bjork,

Basically, any time that you, as a learner, look up an answer or have somebody tell you or show you something that you could, drawing on current cues and your past knowledge, generate instead, you rob yourself of a powerful learning opportunity. (Bjork and Bjork 2011, 61)

In an essay on the need for struggle in learning, Ted Chiang makes a now-popular analogy: “Using ChatGPT to complete assignments is like bringing a forklift into the weight room; you will never improve your cognitive fitness that way” (Chiang 2024). Treating LLMs as Answer Machines prevents learners from retrieving and generating knowledge. We get answers (that may or may not be right), but we're not changed or improved or reshaped through that process.

### Forklifts outside weight rooms

I'm writing this essay at the end of a semester while grading students' final projects. Despite my pleas throughout the course to avoid LLMs—particularly at

the beginning before they know what they're doing—the majority of these assignments use 100% LLM-generated code. Many students have indeed replaced the work of learning with LLM forklifts. As a purely anecdotal observation, students' technical performance is now admittedly high, but the quality of learning is markedly worse than in previous years, with lower quality outputs and shallower understanding.

However, institutional limitations and job-market realities prevent me from banning these tools outright.<sup>3</sup> In industry, computer scientists, engineers, and analysts are now spending substantial time working with LLMs. A June 2025 survey of programmers in the United States reports that 91% of respondents use LLMs for work, with most finding them useful and time-saving (Sleegers and Elsey 2025). The release of agentic tools like Claude Code and OpenAI's Codex in the second half of 2025 shifted the nature of programming work even more dramatically. Analysts and engineers now often “spend their days talking to the A.I., describing in plain English what they want from it and responding to the A.I.'s ‘plan’ for what it will do” (Thompson 2026). Instead of writing actual code, they shepherd, babysit, argue with, and orchestrate LLM agents as they work through large codebases. The rapid adoption of these agentic tools has been staggering: in February 2026, 4% of public commits on GitHub were authored by Claude Code (O'Laughlin et al. 2026), in April 2026, Google reported that 75% of all its new code is AI-generated (Pichai 2026), and Anthropic itself claims that nearly 100% of its code is now created through Claude Code (Zeff 2026). LLMs are also increasingly common in data analytics, and in turn, quantitative social science (Than 2026). And, as I recounted in the introduction of this essay, I myself use these tools.

LLM forklifts obviously have value outside of weight rooms. But that value only comes from existing knowledge and expertise. In my own LLM work creating interactive visualizations, I end up discarding much of what LLMs generate—particularly its styling, colors, and odd LLM-flavored code smells (Bryan 2018)—because I have long experience with CSS, HTML, graphic design, R, and Python. I know enough to recognize which parts of the output are necessary and actually answer my question, and which parts are superfluous, inadequate, or wrong. I can only do this because of my pre-existing, pre-built foundational knowledge.

I've seen the same process with other adult learners when running R trainings for corporations switching from programs like SPSS and SAS to R. Researchers with long experience working with SAS macros for statistical modeling often use LLMs to translate concepts from SAS to R. They know the expected output of their newly translated code and can evaluate whether it works, and they often seek out additional understanding by checking the documentation and asking me (or

---

<sup>3</sup>Google's integration of Gemini-based answers into search results makes this impossible anyway—learners cannot look away from the R code that the LLM provides on demand.

an LLM) to explain why certain functions were used. Again, this kind of work with LLMs is effective because these learners already have pre-existing long-term knowledge.

The world of data engineering and analytics has embraced LLMs, with varying degrees of thought for the epistemic consequences of using an Answer Machine for critical analysis. Some software purports to be able to use AI to discover, explore, model, and write up entire analyses, “freeing your team for high-value work” (OpenAI 2026), which feels reckless. (And isn’t the process of discovering, exploring, modeling, and writing *the* high-value work?) Others seem to approach this with more care. For instance, in 2025, Posit—the company that develops the statistical programs RStudio and Positron—released a new LLM-based tool for analyzing data called Databot that can access live R sessions and generate code that reflects currently-loaded data. In a blog post announcing the release, Posit’s CTO touted its new features, but ended with a remarkable caveat:

In my 30-year career writing software professionally, Databot is both the most exciting software I’ve worked on, and also the most dangerous. (Cheng and Altman 2025)

Cheng’s claim that Databot is dangerous is not due to standard risks of using live data with LLMs (i.e. leaking private or proprietary information). Instead, it’s a danger based in cognitive learning science and pedagogy. He warns that beginners *cannot use it safely*:

...to use Databot effectively and safely, you still need the skills of a data scientist: background and domain knowledge, data analysis expertise, and coding ability. (Cheng and Altman 2025)

There is no LLM-based shortcut to those skills. Following Bjork and Bjork (2011), you cannot LLM your way into durable domain knowledge, data analysis expertise, or coding ability—these only come through struggle. Moreover, those skills can only be used by people who can understand the limitations of the resulting output. Forklifts are incredibly helpful outside the weight room, but only after mastery and practice.

## Maintaining forklift skills

So: no to forklifts in the gym, but yes to forklifts in the real world. The adoption of agentic LLM tools has accelerated quantitative and computational work, but to use these tools well, users must have a strong foundation of knowledge and skills. However, even skilled users need to actively manage how they engage with

these tools to avoid epistemic risks like cognitive surrender (Shaw and Nave 2026) and deskilling (Greengard 2025). To overstretch the running metaphor—forklifts require special training courses and certification, and forklift operators in the United States must renew their licenses every three years (29 CFR §1910.178). The concept of “desirable difficulties” (Bjork and Bjork 2011) does not apply only to learners. I argue that experts should purposely seek out difficulty and introduce friction into their workflow to sharpen their problem-solving skills and maintain a sense of ownership and humanity in their work.

Personally, I find the aesthetics of AI-generated images generally repulsive and I find LLM-generated text to be dull, predictable, and full of odd tics (“it’s not just X — it’s Y”). It lacks a human touch. I enjoy creating things. I like being the human behind all my work. Outsourcing all of the grunt work of writing and analysis to LLMs removes humans from the process and creates—for me, at least—something that lacks ownership. For instance, in an effort to save time, I’ve occasionally attempted to use Claude Code to create software packages from a single LLM prompt. The results seem to work, but I feel disconnected from them and rarely ever revisit them. The output feels unearned. I barely touched these projects, I was hardly involved in their creation, and I don’t fully trust them. There’s no humanity, and I feel no sense of ownership over the code.

In 2012, NASA published an internal report investigating failures in its past missions (Blair et al. 2012; Boykis 2026b). One of their top findings was that outsourcing its engineering to subcontractors led to degraded engineering quality, not because quality of the subcontracted work was poor, but because ownership was diluted (Boykis 2026a). NASA’s internal engineers stopped spending time with “hands-on” work and instead shifted to oversight and monitoring and subsequently lost some of their connection to those processes. Boykis (2026a) argues that “ownership comes from hands on the code and data” and that outsourcing that—whether to NASA subcontractors or LLM agents—reduces that ownership.

LLM-enabled research still must be human driven, with humans making decisions and humans looking at the code and humans understanding the code and humans running the code and humans writing about the output. This is especially important with quantitative scientific work, where researchers are fundamentally responsible for the ownership and validity of their code (Poldrack 2025). While it is possible to unleash swarms of LLM agents that search the literature, design experiments, write IRB protocols, collect data, write results, and even submit papers to journals, I strongly disagree with this approach. Beyond valid arguments that this intense level of LLM use borders on plagiarism and risks swamping the scientific record with slop, humans are absent and disengaged from this workflow, relinquishing their responsibility and ownership in the interest of efficiency.

While LLMs theoretically allow us to remove friction and improve efficiency, I believe that should not be our goal as quantitative scholars. I find myself agreeing with Tim Requarth:

The path forward means acknowledging that efficiency isn't always progress, that removing friction isn't always improvement, and that tools designed to make us more productive might sometimes make us less capable. (Requarth 2025)

Introducing friction back into our workflows—returning to the struggle and desirable difficulties we face when initially learning these skills—can help reclaim humanity and maintain ownership when working with LLMs. My own response to this has been to treat LLM output as inherently provisional and ephemeral and to introduce deliberate friction wherever possible. For example, I approach LLMs with extreme skepticism. I don't use Claude Code in my actual project repositories or give it access to my complete real data. Instead, I use built-in toy datasets or synthetic samples of my data and create small working examples of new analytical techniques I want to use. I use LLMs to help work through this starter code and then apply it to my real work on my own, adapting and recontextualizing the code for my live data. Relatedly, I treat code-related LLM output like answers from online question and answer forums like StackOverflow or blog posts that I have stumbled across—code that is not necessarily true, not peer reviewed, might not even work, but which is potentially a useful demonstration of something I'm trying to do. Occasionally I even ask LLMs to generate their answers and reasoning in the form of a blog post, which makes the output feel more like found artifact I stumbled across rather than an authoritative answer.

Juavinett and Hicks (2026) and Juavinett (2026) call for more deliberate integration of LLMs into learning, but in a way that encourages and deepens understanding rather than improving performance. To help with this, Hicks has created two Claude Code skills based on the metacognitive research of Bjork and Bjork (2011). `learning-goal` (Hicks 2026a) helps users define explicit learning goals and plan out learning intentions, allowing users to “build back a protective habit into [their] day” by scheduling time to deliberately practice new skills. This skill pairs with `learning-opportunities` (Hicks 2026b), which offers users an adaptive lesson after completing important tasks during a Claude Code session. For instance, after creating new files or refactoring code, the skill begins a set of short Bjork-style deliberately difficult learning exercises that encourage generation, spacing, and retrieval based on the user's existing code base. I've personally enabled these skills when translating work from R to Javascript or LaTeX to Typst, and as a result I have a much stronger understanding of how these new-to-me languages work.

Adding friction like this goes against the ongoing push to maximize efficiency and never touch code again. These techniques deliberately slow things down. I have not personally seen huge gains in efficiency and speed in my own work because I'm often spending time translating and restructuring code from a toy example in Claude Code to my real work. However, I have seen substantial gains in my own learning and understanding. I've found that this deliberate friction complements the desirable difficulties I incorporate in my teaching and helps me reconcile my seeming hypocrisy of "AI for me, not (yet) for thee." For both learners and experts, effective LLM use requires developing and maintaining the knowledge that comes from difficulty and struggle.

#### **i** AI usage statement

AI usage statement: I used Anthropic's Claude Sonnet 4.6 for editorial feedback after initial drafting.

## References

- Bastani, Hamsa, Osbert Bastani, Alp Sungu, Haosen Ge, Özge Kabakçı, and Rei Mariman. 2025. "Generative AI Without Guardrails Can Harm Learning: Evidence from High School Mathematics." *Proceedings of the National Academy of Sciences* 122 (26): e2422633122. <https://doi.org/10.1073/pnas.2422633122>.
- Beatty, Sally. 2024. "Khan Academy and Microsoft Partner to Expand Access to AI Tools That Personalize Teaching and Help Make Learning Fun." *Microsoft Source*, May 21. <https://web.archive.org/web/20250325224614/https://news.microsoft.com/source/features/ai/khan-academy-and-microsoft-partner-to-expand-access-to-ai-tools/>.
- Bjork, Elizabeth Ligon, and Robert A. Bjork. 2011. "Making Things Hard on Yourself, but in a Good Way: Creating Desirable Difficulties to Enhance Learning." In *Psychology and the Real World: Essays Illustrating Fundamental Contributions to Society*, edited by Morton Ann Gernsbacher, Richard W. Pew, Leaetta M. Hough, and James R. Pomerantz. Worth Publishers.
- Bjork, Robert A. 1975. "Retrieval as a Memory Modifier." In *Information Processing and Cognition: The Loyola Symposium*, edited by Robert L. Solso. Erlbaum.
- Blair, J. C., R. S. Ryan, and L. A. Schutzenhofer. 2012. *Elements of Engineering Excellence*. NASA/CR—2012-217471. NASA. <https://ntrs.nasa.gov/api/citations/20130000445/downloads/20130000445.pdf>.
- Boykis, Vicki. 2026a. "Build Yourself Flowers." April 20. <https://web.archive.org/web/20260426035611/https://vickiboykis.com/2026/04/20/build-yourself-flowers/>.
- Boykis, Vicki. 2026b. "NASA Elements of Engineering Excellence." April 5. <https://web.archive.org/web/20260430181213/https://vickiboykis.com/2026/04/05/nasa-elements-of-engineering-excellence/>.

- Bryan, Jenny. 2018. “Code Smells and Feels.” userR!2018, Brisbane, July. <https://github.com/jennybc/code-smells-and-feels>.
- Cheng, Joe, and Sara Altman. 2025. “Databot Is Not a Flotation Device.” Posit, August 25. <https://web.archive.org/web/20250910084327/https://posit.co/blog/databot-is-not-a-flotation-device/>.
- Chiang, Ted. 2024. “Why A.I. Isn’t Going to Make Art.” *The New Yorker*, August 31. <https://www.newyorker.com/culture/the-weekend-essay/why-ai-isnt-going-to-make-art>.
- Greengard, Samuel. 2025. “The AI Deskillling Paradox.” *Communications of the ACM*, November 7. <https://web.archive.org/web/20251110043905/https://cacm.acm.org/news/the-ai-deskillling-paradox/>.
- Hicks, Cat. 2026a. *Learning Goal: A Claude Code Skill for Structured Goal Setting with Mental Contrasting*. Released April 21. <https://github.com/DrCatHicks/learning-goal>.
- Hicks, Cat. 2026b. *Learning Opportunities: A Claude Code Skill for Deliberate Skill Development*. Released April 22. <https://github.com/DrCatHicks/learning-opportunities>.
- Juavinett, Ashley. 2026. “How to Teach Programming in the Age of AI.” *The Transmitter*, ahead of print. <https://doi.org/10.53053/IXOZ2506>.
- Juavinett, Ashley, and Catherine M. Hicks. 2026. *You Can Learn with AI*. Season 2, episode 4, February 16. <https://www.buzzsprout.com/2396236/episodes/18692591-you-can-learn-with-ai>.
- Lehmann, Matthias, Philipp B. Cornelius, and Fabian J. Sting. 2025. “AI Meets the Classroom: When Do Large Language Models Harm Learning?” Pre-published March 8. <https://doi.org/10.48550/arXiv.2409.09047>.
- O’Laughlin, Doug, Jeremie Eliahou Ontiveros, Jordan Nanos, Dylan Patel, and Daniel Nishball. 2026. “Claude Code Is the Inflection Point.” *SemiAnalysis*, February 5. <https://web.archive.org/web/20260430141617/https://newsletter.semianalysis.com/p/claude-code-is-the-inflection-point>.
- OpenAI. 2026. “ChatGPT for Data Science & Analytics.” <https://web.archive.org/web/20260330095922/https://chatgpt.com/business/ai-for-data-science-analytics/>.
- Pichai, Sundar. 2026. “Cloud Next ‘26: Momentum and Innovation at Google Scale.” Google, April 22. <https://web.archive.org/web/20260425012108/https://blog.google/innovation-and-ai/infrastructure-and-cloud/google-cloud/cloud-next-2026-sundar-pichai/>.
- Poldrack, Russell. 2025. “AI-assisted Coding: 10 Simple Rules to Maintain Scientific Rigor.” *The Transmitter*, ahead of print. <https://doi.org/10.53053/LCDN3424>.
- Requarth, Tim. 2025. “From Bench to Bot: Why AI-powered Writing May Not Deliver on Its Promise.” *The Transmitter*, ahead of print. <https://doi.org/10.53053/HZQR1694>.
- Root, William B., and Ruth Anne Rehfeldt. 2021. “Towards a Modern-Day Teaching Machine: The Synthesis of Programmed Instruction and Online Education.” *The Psychological Record* 71 (1): 85–94. <https://doi.org/10.1007/s40732-020-00415-0>.
- Shaw, Steven D, and Gideon Nave. 2026. “Thinking—Fast, Slow, and Artificial: How AI Is Reshaping Human Reasoning and the Rise of Cognitive Surrender.” Pre-published January 12. [https://doi.org/10.31234/osf.io/yk25n\\_v1](https://doi.org/10.31234/osf.io/yk25n_v1).
- Shen, Judy Hanwen, and Alex Tamkin. 2026. “How AI Impacts Skill Formation.” Pre-published February 1. <https://doi.org/10.48550/arXiv.2601.20245>.

- Sleegers, Willem, and Jamie Elsey. 2025. *Adoption and Uses of LLMs Among U.S. Tech Workers*. Rethink Priorities. <https://rethinkpriorities.org/wp-content/uploads/2025/07/LLM-industry-survey.pdf>.
- Soderstrom, Nicholas C., and Robert A. Bjork. 2015. "Learning Versus Performance: An Integrative Review." *Perspectives on Psychological Science* 10 (2): 176–99. <https://doi.org/10.1177/1745691615569000>.
- Stewart, Rebecca. 2025. "Hi! It's Duo: The Marketing Strategy Behind the Internet's Favorite Green Menace." *Adweek*, April 8. <https://web.archive.org/web/20250425210150/https://www.adweek.com/brand-marketing/duolingo-duo-owl-marketing-strategy/>.
- Than, Ker. 2026. "Social Scientists Embrace the AI Moment." Stanford Report, April 24. <https://news.stanford.edu/stories/2026/04/ai-social-science-empirical-research>.
- Thompson, Clive. 2026. "Coding After Coders: The End of Computer Programming as We Know It." *The New York Times Magazine*, March 12. <https://www.nytimes.com/2026/03/12/magazine/ai-coding-programming-jobs-claude-chatgpt.html>.
- Twyman, Janet S. 2020. "Programmed Instruction." In *The Encyclopedia of Child and Adolescent Development*, edited by Stephen Hupp and Jeremy Jewell. Wiley. <https://doi.org/10.1002/9781119171492.wecad095>.
- Watters, Audrey. 2021. *Teaching Machines: The History of Personalized Learning*. The MIT Press. <https://doi.org/10.7551/mitpress/12262.001.0001>.
- Zeff, Maxwell. 2026. "How Claude Code Is Reshaping Software—and Anthropic." *Wired*, January 22. <https://www.wired.com/story/claude-code-success-anthropic-business-model/>.