Executive Summary

Recent estimates suggest the midpoint for automating half of current work activities could arrive as early as 2030, and even conservative scenarios now track a slope similar to what was once the optimistic projection. The standard response, that we should retrain workers, runs into a well-documented problem: knowledge acquired in one context rarely transfers to another, and adult learning infrastructure barely exists outside of compliance training.

What does work is cognitive apprenticeship: learning embedded in real practice, with an expert present to scaffold at the right moment. But apprenticeship requires co-presence with a knowledgeable person, which means it depends on geographic and social luck and cannot scale. How will we retrain large swarths of the population for jobs that might not even exist today?

In this essay, I argue that embedded learning systems offer a way to approximate apprenticeship at scale. Rather than pulling learners into separate tutoring interfaces, these systems observe work as it happens, maintain a model of the learner's developing competence, and provide scaffolding tuned to the moment. The core components involve screen-level context understanding, persistent user models, and adaptive intervention, all already exist as research contributions that require refinement. The challenge is integration, and ensuring the pedagogy remains open, auditable, and not controlled by a handful of companies.

The Transfer Problem

In 2018 I tried to teach myself to code the way everyone tells you to teach yourself to code. I enrolled in online classes, watched lectures on data structures, and completed some exercises on recursion. After each course, I did well on quizzes and would feel briefly competent. Then I'd try to build something of my own, and the knowledge would evaporate. Three courses in, I had little to show for it other than some checkmarks on Coursera.

What eventually worked was not 'educational'. It was moving to San Francisco and falling into a group of software engineers. Nobody sat me down and taught me anything, but suddenly I was around it. Watching them debug side projects in real time, hearing how they describe their work, picking up on what "good" looked like before I could even articulate why. By the end of the summer, something had shifted. "Programmer" stopped being an aspiration and started feeling like a description.

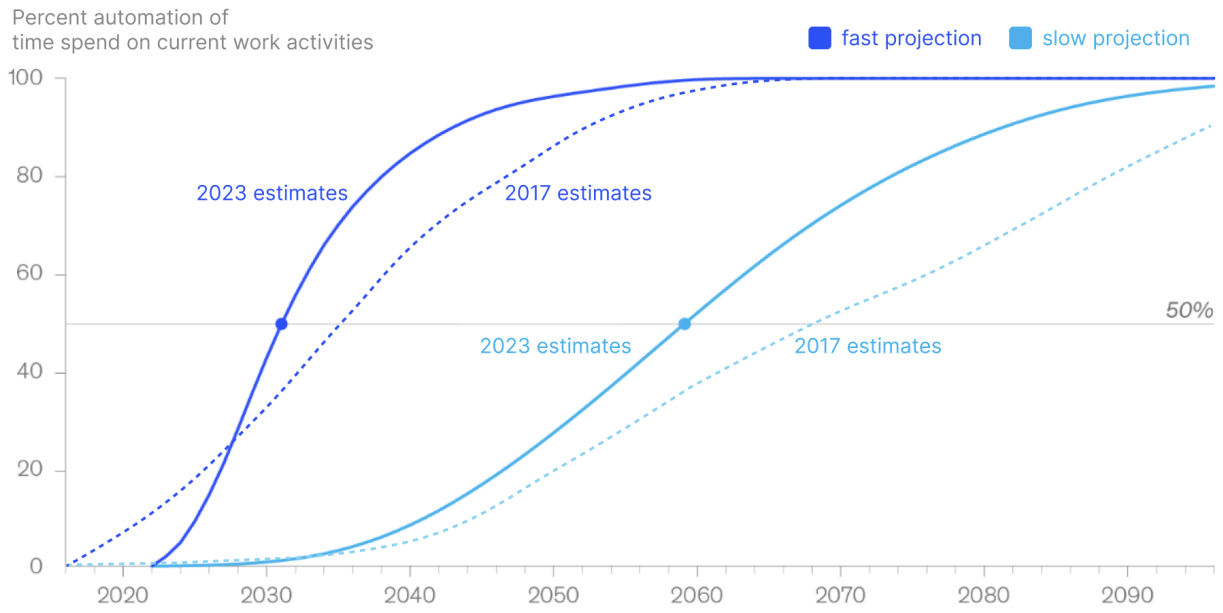
Learning scientists have a name for the thing that didn't happen during my online courses: the transfer problem[1]. Knowledge acquired in one context often fails to transfer to another. You learn recursion in a browser tab, and it stays in the browser tab. Estimates of how often transfer actually occurs vary by subject, but they're often grim with some as low as 10%[2]. But this isn't a mystery, though. Research in cognitive science has shown that memory is context-dependent [3]. You encode not just the content but the situation. When the situation swaps from your browser to your code editor, retrieval fails.

I bring this up now because we're about to need transfer to work at a scale it has never worked before.

---

[1] Packer, M. (2001). The Problem of Transfer, and the Sociocultural Critique of Schooling. Journal of the Learning Sciences.
[2] Georgenson, David L. (1982). The Problem of Transfer Calls for Partnership. Training and Development Journal, 36, 75–78]
[3] Tulving, Endel. Thomson Donald. (1973) Encoding specificity and retrieval processes in episodic memory. Psychological Review.

The scale of displacement



Source: Modified from McKinsey's *The economic potential of generative AI: The next productivity frontier report*. Exhibit 8

Look at Figure 1. It shows McKinsey's estimates of how much of current work could be automated, and how quickly. There are four curves: two scenarios (fast adoption and slow adoption), each estimated twice (in 2017 and again in 2023, after generative AI). Every curve shifted left. In the fast scenario, the 50% midpoint jumps to 2030. But the more striking thing is what happened to the *slow* scenario. The 2017 slow estimate barely reaches 50% by 2070. The 2023 slow estimate, the *conservative* projection, the one where adoption drags its feet now shares a similar slope to the 2017 *fast* scenario, even if offset by two decades.

What was supposed to be a multi-generational transition now looks like it could happen within the span of a single career, even if everything goes slowly.

This doesn't mean half of all jobs disappear by 2030. It means the *contents* of jobs are changing faster than people can update their skills to match. McKinsey estimates[4] many Americans will need to switch occupational categories entirely. The standard response to this kind of displacement is always the same: retrain them.

But retrain them through what, exactly?

According to the OECD's 2025 Survey of Adult Skills[5], the dominant form of adult learning across member countries is short, employer-provided, compliance-based training (think anti-harassment modules or safety certifications). This is training designed to check a legal box,

[4] Kweilin Ellingrud et Al (2023). Generative AI and the future of work in America.
[5] OECD (2025). Trends in Adult Learning.
https://www.oecd.org/content/dam/oecd/en/publications/reports/2025/06/trends-in-adult-learning_fod8514f/ec0624a6-en.pdf

not to build new competencies. Formal adult education, the kind that might actually reskill someone for a different occupation, accounts for just 8% of adult learners and is declining. Formal education can't absorb the gap.

We're staring at a situation where hundreds of millions of people need to acquire new competencies, and the primary tool we have for delivering that learning is the one that already doesn't work. Making the courses better isn't the fix.

The Apprenticeship Problem

There's a long-running argument in learning science that maps neatly onto my failed online courses.

On one side are the constructivists. You build understanding through real problems and mistakes. On the other side, the instructivists. They argue that novices don't know what they don't know. If left to explore, they flounder. It is better to show them how experts do it through worked examples, explicit instruction, and managed cognitive load.

What would be ideal need is both at once. Real problems with adaptive scaffolding. Authentic engagement with a safety net. For most of history, the only technology that could deliver this was a person: a master watching an apprentice, intervening at the right moment with the right kind of help. Researchers call this Cognitive apprenticeship[6], and it works beautifully. It also requires winning a geographic and social lottery.

I often think of the time I lived in San Francisco as one of the highest growth experiences of my life. I didn't know any of this terminology at the time, but cognitive apprenticeship is what I stumbled into, and the reason it worked is the reason it can't scale. It depended entirely on the accident of who I happened to be around and the fact that they were willing to give their time.

You can't give millions of displaced workers a friend group full of senior engineers.

Or can you?

[6] Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L. B. Resnick (Ed.), Knowing, learning, and instruction: Essays in honor of Robert Glaser (pp. 453–494).

Embedded Learning Systems

Here's what it looks like when most people use AI to learn something right now. You're working on a spreadsheet and you get stuck. You open a new tab, navigate to ChatGPT, and try to describe what you were doing.

Most people currently view AI as a tutor in a chat window. But it reproduces exactly a similar problem we started with: you extract yourself from the context where you need the knowledge in order to go ask about it. The learning is decontextualized by default.

It's also unrealistically tidy. Real knowledge work doesn't happen in one application. A data science student might start in a spreadsheet, write a query in a SQL client, paste results into a notebook, discuss findings in Slack, and return to the spreadsheet. The *task* is one continuous thread. The tools are five separate worlds that know nothing about each other. And each AI assistant (Copilot in the code editor or Gemini in the Google Doc) only sees its own keyhole. Nobody sees the task. Such an issue led to the invention of xAPI, a standard for collecting learning data across platforms.

Beyond instrumentation, cognitive apprenticeship requires *co-presence*. The master in the workshop doesn't review a transcript of the apprentice's mistakes at the end of the day. They're standing next to the apprentice *while the mistakes are happening*, and they intervene at the moment intervention is useful. That's the difference between a system that records your learning and a system that participates in it.

Another way to put it: pull-based learning is when you go looking for help because you know you have a gap, and push-based learning, where the system can see the gap and offer support based on what it observes. Pull-based learning requires you to know what you don't know, which is precisely the thing novices are worst at. Push-based learning removes that requirement. The system watches, models your developing competence, and scaffolds at the edge of what you can currently do.

The pieces for this already exist,. Modern AI models can interpret screen content and reason about user intent [7]. And researchers in user modeling have spent decades developing what Judy Kay and Bob Kummerfeld call General User Models: persistent representations of a user's knowledge, preferences, and goals that follow them across tools. What's new is that these threads are converging. A system that observes your work across applications, maintains a model of your developing expertise, and provides scaffolding tuned to your current edge of competence is not science fiction. It's an engineering problem.

[7] Shaikh, Omar et Al. (2025) Creating General User Models from Computer Use. UIST 2025.

A Case Study on Embedded Learning

Renatta works for a county health department in New Mexico. Her director asked her to prepare a brief for next month's board meeting on environmental health trends, benchmarking their county against state-level patterns. A colleague pointed her to a recent paper with a similar analysis dataset and the notebook used to generate the published figures. "Start from their analysis," her colleague said. "Adapt it to our question."

Renatta downloads the dataset and opens it in Excel. She knows Excel. She's built pivot tables, made charts for grant reports, filtered and sorted her way through messy spreadsheets for years. She starts trying to reproduce the paper's map. Excel has a built-in map chart, but it won't let her control the color scale the way the paper does, and she can't figure out how to normalize the values before plotting. After twenty minutes she has something that looks nothing like the published figure.

She opens the paper's notebook file out of curiosity. Code cells instead of a grid. No visible rows and columns. She can see the output (the same map from the paper) clearly, but the connection between the code and what it produces is opaque. She tries changing a number in one of the cells and reruns it. Something happens, but she's not sure what.

An embedded learning system is present in the background, bundled with the notebook environment. It can see what Renatta sees: the notebook, the data, the sequence of actions. It does not generate answers. It models the context she's working in.

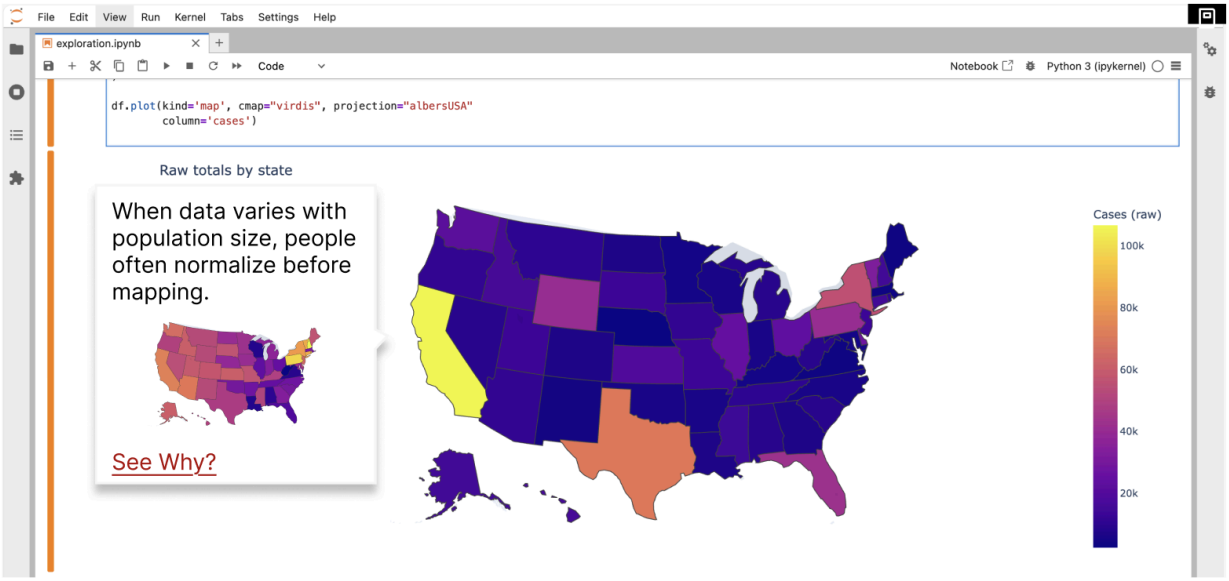
The system notices Renatta scrolling between a code cell and its output repeatedly. It surfaces a short annotation next to the cell: "This block does something similar to a pivot table. It groups the data by state and sums the values in the 'cases' column."

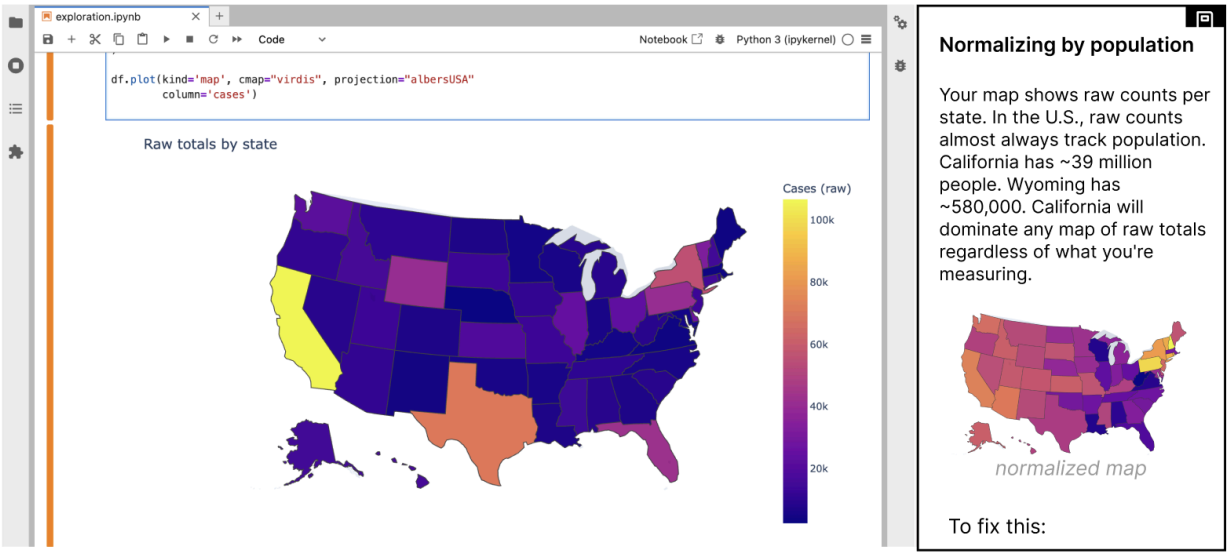
That small bridge is enough for her as Renatta begins modifying rather than just reading. The paper's notebook already has code that produces a state-level map so she then swaps in her own dataset's column names, reruns the cell, and gets a result. It's wrong, but it's hers. She tweaks a color scale parameter she saw in the original and sees the map change. She's not writing code yet. She's editing someone else's code the way she might edit someone else's spreadsheet, changing values to see what happens. The notebook is still mostly opaque, but the feedback loop is fast enough to keep her going.

Her first real map aggregates raw counts by state. California immediately dominates the visualization, dwarfing everything else. Renatta tweaks color scales, changes projections, but nothing helps.

After a few more attempts, the system surfaces a gentle suggestion: "When data varies with population size, people often normalize before mapping. Want to see why?"

Renatta clicks.





Her notebook stays the same. No cells re-run. But next to her notebook, a side panel opens explaining how when using raw counts, states with more people are likely to have higher case counts. It recommends a specific edit to her code and offers a space for follow-up questions.

Renatta makes the fix. Her map changes. She moves on.

Later, she's working on a scatterplot comparing two variables from her dataset. The result looks noisy and flat. She adjusts axis limits, reruns the cell. Still nothing useful. The chart just looks like a cloud.

She clicks the embedded learning system button in the top right corner of her screen, a persistent toggle that tells the system she has a question. Then she navigates to the paper she's trying to reproduce, shift-clicks the original figure to select it, switches back to her

notebook and clicks her own chart. Both artifacts now appear in the learning panel's input field. She types "why are these different" and hits send.

The system pulls in the paper's figure, its caption, the methods section describing how it was generated. It already has Renatta's notebook context: her data, her code, the sequence of things she's tried.

In the learning panel, the system puts the two charts side by side and annotates three differences. First: the paper's x-axis is log-scaled. Renatta's is linear, which is why her cloud is compressed into the bottom-left corner while the paper's points spread across the full plot. Second: the paper filters to counties with more than 10,000 residents, which Renatta's version doesn't. The tiny counties are adding noise without adding signal. The system shows what Renatta's own scatterplot looks like with and without these changes, using her data, so she can see the trend emerge from the cloud she was staring at.

Below the comparison, a short note: "The original authors made at least three choices that aren't visible in the final figure. Most published charts look clean because of decisions like these, not because the raw data was clean."

As Renatta continues, the system adapts. When she repeatedly copies and modifies code, it infers trial-and-error rather than hypothesis testing. When she lingers on one view, it surfaces alternatives without insisting.

What Renatta is learning is not just how to use a notebook. It's how to think about data the way an analyst does: which transformations reveal structure, which visualizations mislead, which choices are invisible in a finished chart.

What could go Wrong

First is the Clippy problem. Microsoft's Clippy tried to be contextually helpful and became reliably irritating. Systems that intervene too often will be disabled. Systems that intervene too rarely won't produce learning.

But the optimal intervention policy varies not just across people but across moments for the same person. Deep focus makes any interruption costly. Frustration makes a well-timed suggestion a gift. Playful exploration means staying quiet and letting productive mistakes happen. The system needs to read these states through behavioral signals like pause patterns, error rates, and task-switching frequency. This is a research problem and is not a fully solved one. Though existing work has indicated that modern Vision Language Models tend to infer user context well[8]. Kay's work[9] on scrutable user models point toward one solution: giving learners explicit control over when and how the system intervenes, so the scaffolding adapts to the person rather than the other way around.

[8] Michelle, Lam. (2025) Just-In-Time Objectives: A General Approach for Specialized. AI Interactions.
[9] Judy Kay, Bob Kummerfeld (2019) From data to personal user models for life-long, life-wide learners. British Journal of Educational Technology. Volume 50, Issue 6 pp. 2871-2884

Secondly is social erosion. My San Francisco story was not just about contextual learning. It was about identity formation, social accountability, and the organic unpredictability of watching real humans navigate real problems. If these systems become the primary source of on-the-job learning support, they may reduce the incentive to ask colleagues for help. In doing so it may erode the socializing that builds comradery between coworkers. Learning from other people builds relationships. Learning from an AI does not.

Furthermore, digital scaffolding is inherently limited. A coding environment has a property that makes digital scaffolding natural: the work *is* the screen. But a nursing student practicing intubation or a diver learning buoyancy control involve skills that are embodied. We don't have good computational representations for most of what matters in physical practice. You can watch a video on how to hold neutral buoyancy at a safety stop, but the actual skill is in your breath. The actual skill is fully embodied. Such as how you feel if you're rising or sinking before your depth gauge changes. No screen captures that.

Who decides what to learn?

If the system decides what gap you have, when to intervene, and what to teach you, who's writing the curriculum? In traditional education, there are at least legible structures of accountability: syllabi, accreditation boards, teachers you can argue with. A system that's watching your work and nudging your learning is making pedagogical decisions constantly, and those decisions are shaped by whoever built the model, trained it, and defined what "competent" looks like.

This is why the model layer matters as much as the interface layer. If embedded learning runs exclusively on proprietary models controlled by a handful of companies, then the pedagogy is proprietary too. It becomes opaque, unauditable, and shaped by commercial interests that may or may not align with the learner's. We need a healthy diversity here: publicly funded learning models trained on open educational data, open-source alternatives that researchers and educators can inspect and modify, and enough competition that no single company gets to define what "learning" means inside the tools where most people work.

Would it have worked for me?

The question I can't quite resolve is whether any of this would have worked for me.

Whether an AI can deliver the existential shift like what I experienced in San Francisco, feels like a open question. I am not certain it can, but also, I’m not certain it needs to work the same way. Consider someone who uses an AI-embedded coding environment for six months and gradually starts thinking of themselves as someone who builds software. This identity shift is not only because of social mirroring, because they’re actually producing things that work. Identity shift might be bootstrapped by their accumulated evidence of competence.

Education is the sector where AI can do the most good because it is the sector that makes every other sector better. The infrastructure we build for learning determines the pace at which people can adapt to a changing world.