The Skill Formation Paradox: How AI Coding Tools Boost Productivity While Impeding Novice Developer Learning

Anonymous Author(s)

ABSTRACT

AI coding assistants provide substantial productivity gains to novice software developers, yet their impact on underlying skill formation remains an open question with significant implications for the software engineering workforce. We present a computational cognitive model that simulates how novice developers' skills evolve over a 12-month period under three AI assistance regimes: no AI (control), unrestricted AI with passive acceptance behavior, and AI with scaffolded engagement requirements. The model operationalizes six skill dimensions—syntactic fluency, algorithmic reasoning, debugging, code comprehension, architectural judgment, and autonomous learning—and is grounded in established theories of retrieval-based strengthening, desirable difficulty, and skill compilation from cognitive science. Our simulation of 240 developers (80 per condition) over 252 working days reveals a *skill formation paradox*: unrestricted AI use produces a large negative effect on skill development (Cohen's $d = -1.04$), with the strongest impairment in highly automatable skills such as syntactic fluency ($d = -5.10$), while scaffolded engagement nearly eliminates this deficit ($d = -0.04$ overall). Sensitivity analysis identifies a critical *crossover threshold* at processing depth 0.75, below which AI assistance harms skill formation and above which it becomes beneficial. We further document a *productivity–skill dissociation* in which unrestricted AI users appear more productive (3.69 vs. 3.21 tasks/day) yet possess weaker underlying skills (0.56 vs. 0.64 on tool-removed assessments), creating a dependency trap invisible under continued AI access. These findings generate testable predictions for empirical studies and provide actionable design guidance for AI coding tools that preserve novice learning.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Computing methodologies** → **Modeling and simulation**; • **Software and its engineering** → *Software development techniques.*

KEYWORDS

AI coding tools, skill formation, novice developers, cognitive modeling, scaffolded learning, productivity paradox

ACM Reference Format:

Anonymous Author(s). 2026. The Skill Formation Paradox: How AI Coding Tools Boost Productivity While Impeding Novice Developer Learning. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '26), August 3–7, 2026, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

KDD '26, August 3–7, 2026, Toronto, ON, Canada

ACM ISBN 978-1-4503-XXXX-X/26/08…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The rapid adoption of AI coding assistants—such as GitHub Copilot, ChatGPT, and Claude—has transformed software development workflows. Empirical evidence demonstrates that these tools yield substantial productivity gains, particularly for less experienced developers [7, 13, 15]. Shen et al. [15] document that junior developers experience disproportionately large speed improvements when using AI assistance, a finding consistent with earlier controlled studies [13].

However, productivity and skill are distinct constructs. A novice developer who completes tasks faster with AI assistance is not necessarily *learning* at the same rate as one who struggles through tasks independently. Shen et al. [15] explicitly identify this gap, noting that "the effect of these tools on the skill formation of this subgroup remains unknown." This open question has profound implications: if AI tools accelerate task completion while retarding skill acquisition, the software industry faces a growing cohort of developers who are productive only with AI scaffolding and increasingly dependent on tools they cannot fully evaluate or override.

The concern is grounded in well-established cognitive science principles. Retrieval-based strengthening theory [4] holds that skills consolidate through active recall and application; AI tools that provide ready-made solutions may bypass this retrieval process. The desirable difficulty framework [3] demonstrates that moderate challenge during practice enhances long-term retention, even at the cost of immediate performance—precisely the trade-off that AI assistance reconfigures. Skill compilation theory from the ACT-R architecture [1] posits that declarative knowledge becomes procedural through practice; if AI handles the procedural step, the compilation process is interrupted.

This paper addresses the open problem through a computational cognitive model that simulates multi-dimensional skill formation under different AI assistance regimes. Our contributions are:

(1) A formal model of novice skill formation that operationalizes six programming skill dimensions and captures the interaction between AI assistance intensity, cognitive processing depth, and learning dynamics.
(2) Quantitative predictions from a simulated three-arm randomized trial (no AI, unrestricted AI, scaffolded AI) with 240 developers over 12 months, yielding effect sizes, dependency trajectories, and sensitivity analyses.
(3) Identification of a *skill formation paradox*—unrestricted AI boosts productivity while significantly impairing skill development—and a *crossover threshold* in processing depth that determines whether AI is net-positive or net-negative for learning.

(4) Actionable design implications for AI coding tools and educational interventions that preserve novice learning.

1.1 Related Work

AI Tools and Developer Productivity. Multiple studies establish that AI coding assistants increase developer throughput. Peng et al. [13] report a 55.8% faster task completion rate with GitHub Copilot in a controlled experiment. Hou et al. [7] find productivity gains across three field experiments, with larger effects for less experienced developers. Shen et al. [15] provide a comprehensive analysis showing that junior developers benefit disproportionately, but explicitly flag skill formation as an unresolved question.

AI and Learning in Educational Contexts. Bastani et al. [2] demonstrate that access to GPT-4 in a mathematics tutoring context harms learning outcomes, providing direct evidence that AI assistance can impede skill acquisition. Kazemitabaar et al. [9] study novice programmers using AI code generators and find mixed effects on learning, with benefits dependent on how students engage with the generated code. Denny et al. [6] survey the landscape of computing education in the generative AI era, identifying the need for pedagogical frameworks that leverage AI while preserving learning. Prather et al. [14] document a widening gap between novice and expert developers when AI assistance is available, raising concerns about differential skill development.

Cognitive Foundations. The desirable difficulty framework [3] and retrieval practice research [4] provide the theoretical basis for predicting that reducing task difficulty through AI assistance may impair long-term learning. The expertise reversal effect [8] suggests that scaffolding beneficial for novices may become counterproductive as expertise develops. Anderson's ACT-R theory [1] models how procedural skills are acquired through practice, offering a formal framework for reasoning about how AI intervention in the practice process affects skill compilation. The Knowledge-Learning-Instruction framework [10] provides additional theoretical grounding for understanding how instructional interventions interact with learning processes.

Human–AI Interaction in Programming. Vaithilingam et al. [16] evaluate the usability of AI code generation tools and find that developers often accept suggestions without deep understanding. Mozannar et al. [11] model user behavior during AI-assisted programming, characterizing the spectrum from passive acceptance to active engagement. Parasuraman and Riley [12] provide the foundational framework on automation use, misuse, and skill degradation—the "automation complacency" phenomenon that may manifest in AI-assisted coding. Weber et al. [17] and Cui et al. [5] examine the broader impacts of AI tools on software engineering tasks and help-seeking behavior, respectively, contributing to our understanding of how AI tools alter the learning environment.

Gap Addressed. While prior work establishes productivity effects and raises learning concerns, no existing study provides a formal model that (a) decomposes programming skill into distinct dimensions, (b) models the interaction between AI assistance intensity and cognitive engagement, and (c) generates quantitative predictions for longitudinal skill trajectories under different AI use regimes. Our computational approach fills this gap and provides a bridge between cognitive theory and empirical study design.

2 METHODS

2.1 Model Overview

We develop a computational cognitive model of skill formation that simulates how novice developers' programming abilities evolve over time under different AI assistance conditions. The model represents each developer as a vector of skill levels across six dimensions, updated daily through task-driven learning dynamics. Three experimental conditions are simulated: **Control** (no AI), **Unrestricted AI** (full AI access with passive acceptance behavior), and **Scaffolded AI** (AI access with mandatory engagement: developers must read, modify, and explain AI-generated code before proceeding).

2.2 Skill Dimensions

Programming competence is operationalized as a six-dimensional skill vector $\mathbf{s} \in [0, 1]^6$:

(1) **Syntactic fluency**: ability to write correct code from specifications without reference materials.
(2) **Algorithmic reasoning**: capacity to solve novel computational problems.
(3) **Debugging**: skill at locating and fixing defects in unfamiliar code.
(4) **Code comprehension**: ability to read, understand, and predict the behavior of code.
(5) **Architectural judgment**: capacity to evaluate and design system-level structures.
(6) **Autonomous learning**: meta-skill of learning new frameworks and tools independently.

Each dimension has a corresponding AI *automation weight* $w_i \in [0, 1]$ reflecting how effectively current AI tools can assist with that skill type. We set $w = (0.80, 0.50, 0.35, 0.25, 0.15, 0.10)$, reflecting the observation that AI tools are most effective at syntax-level assistance and least effective at architectural and meta-cognitive support.

2.3 Task-Driven Learning Dynamics

Each simulated working day, a developer encounters $T = 5$ coding tasks. Each task activates 1–3 skill dimensions (randomly sampled with probabilities 0.4, 0.4, 0.2) and has a difficulty $\delta \sim \mathcal{N}(0.45, 0.15^2)$ clipped to $[0.05, 0.95]$.

Success Probability. The probability of successfully completing a task component in dimension i is modeled as a logistic function:

$$P(\text{success}) = \sigma\big(k \cdot (s_i - \delta_{\text{eff}})\big) \tag{1}$$

where σ is the sigmoid function, $k = 8$ controls steepness, s_i is current skill in dimension i, and δ_{eff} is the effective difficulty (reduced by AI in treatment conditions).

AI Modulation. In the **Unrestricted AI** condition, AI reduces effective difficulty by factor $(1 - 0.55 \cdot w_i)$ and cognitive processing depth to $0.15 + 0.85 \cdot (1 - w_i)$. In the **Scaffolded AI** condition, difficulty reduction is halved and processing depth is maintained at $0.70 + 0.30 \cdot (1 - 0.3w_i)$.

Learning Signal. The learning signal from each task attempt integrates three factors:

$$\ell = D(\delta, s_i) \cdot F(\text{success}, \delta - s_i) \cdot \phi \quad (2)$$

where D captures *desirable difficulty* (a Gaussian centered at gap = 0.10, reflecting optimal learning when tasks are slightly above current skill), F is a success/failure modulator (successful attempts yield factor 0.8; near-miss failures yield 0.4; distant failures yield 0.1), and ϕ is the processing depth.

Skill Update with Transfer. Raw learning signals are transformed through a transfer matrix $\mathbf{T}$ that captures cross-dimensional learning transfer (e.g., improvement in algorithmic reasoning partially transfers to debugging). Skills update as:

$$\mathbf{s} \leftarrow \mathbf{s} + \alpha \cdot (\boldsymbol{\ell} \cdot \mathbf{T}) - \beta \cdot \mathbf{m} \odot \mathbf{s} \quad (3)$$

where $\alpha = 0.006$ is the learning rate, $\beta = 0.0005$ is the forgetting rate, and $\mathbf{m}$ is a binary mask indicating dimensions *not* exercised in the current task (implementing use-it-or-lose-it decay).

2.4 Experimental Design

We simulate a three-arm parallel design with $n = 80$ developers per condition, over $D = 252$ working days (approximately 12 calendar months). Initial skill levels are sampled from $\mathcal{N}(0.20, 0.05^2)$ clipped to $[0.05, 1.0]$, representing novice developers with 0–2 years of experience.

Assessment Protocol. Tool-removed skill assessments are conducted monthly (every 21 working days), yielding 12 assessment time points. Assessment scores equal the true skill level plus Gaussian noise $\mathcal{N}(0, 0.03^2)$, simulating measurement error.

Outcome Measures. Primary outcomes include: (1) *Skill growth*: change in tool-removed skill level from first to last assessment; (2) *Effect sizes*: Cohen's d between conditions at final assessment; (3) *Dependency index*: DI = (AI-assisted − unassisted)/AI-assisted performance; (4) *Productivity*: tasks completed per day with and without AI. Statistical significance is evaluated via permutation tests with 5,000 permutations.

Sensitivity Analysis. We systematically vary the processing depth parameter ϕ from 0.05 to 0.95 (in steps of 0.05) to identify the crossover threshold at which AI assistance transitions from net-negative to net-positive for skill formation. This analysis uses 40 developers per condition to maintain computational efficiency.

3 RESULTS

3.1 Overall Skill Formation

Table 1 summarizes skill trajectories across conditions. All three groups begin with comparable skill levels (≈ 0.23). After 12 months, the Control group reaches a mean skill of 0.643, the Unrestricted AI group reaches 0.562, and the Scaffolded AI group reaches 0.641. The Unrestricted AI condition produces 17.3% less skill growth than Control, while Scaffolded AI produces growth nearly identical to Control.

The overall Cohen's d between Unrestricted AI and Control is −1.04 (large negative effect), indicating that unrestricted AI use significantly impairs skill development. The Scaffolded AI vs. Control effect size is $d = -0.04$ (negligible), indicating that scaffolded engagement preserves nearly all of the learning benefit of unaided practice.

Table 1: Overall skill trajectories by condition. All values are mean skill levels on tool-removed assessments (scale 0–1). Growth is the difference between final and initial assessments.

Condition	Initial	Final	Growth
Control (No AI)	0.238	0.643	+0.404
Unrestricted AI	0.228	0.562	+0.334
Scaffolded AI	0.236	0.641	+0.405

3.2 Dimension-Specific Effects

Figure 1 displays skill trajectories for each of the six dimensions across all three conditions. The magnitude of AI's negative effect is strongly correlated with the dimension's automation weight.

Table 2 reports the dimension-specific final skill levels and effect sizes. Syntactic fluency shows the largest impairment under unrestricted AI ($d = -5.10$, $p < 0.001$), followed by algorithmic reasoning ($d = -2.07$, $p < 0.001$). Architectural judgment shows the smallest effect ($d = -0.44$, $p = 0.006$), consistent with AI tools providing less assistance for high-level design decisions. Under Scaffolded AI, most dimensions show small or non-significant effects relative to Control, with algorithmic reasoning actually showing a small positive effect ($d = +0.34$, $p = 0.031$), suggesting that scaffolded AI engagement may enhance certain reasoning skills.

Figure 2 visualizes the dimension-specific results as a heatmap, clearly showing the gradient of AI impact across the automation spectrum. The Spearman correlation between automation weight w_i and Unrestricted AI effect size is $\rho = -0.94$, confirming that AI most impairs skills in dimensions where it provides the most assistance.

3.3 The Productivity–Skill Dissociation

Figure 3 illustrates the central paradox: unrestricted AI users appear *more* productive when measured with AI access (3.69 tasks/day vs. 3.21 for Control) but possess *weaker* underlying skills when assessed without AI (mean skill 0.562 vs. 0.643).

This dissociation has practical implications: organizations evaluating developer performance based on AI-assisted output metrics will systematically overestimate the capability of developers who rely heavily on AI tools. The gap between measured productivity and genuine skill represents a *hidden dependency* that only becomes visible when AI access is removed or when developers face novel problems outside AI's competence.

3.4 Dependency Index

Figure 4 tracks the Dependency Index (DI) over time. Both AI conditions begin with high DI values (≈ 0.62) due to novice-level starting skills. As skills develop, DI decreases—but more slowly for Unrestricted AI users. At month 12, the Unrestricted AI group retains a DI of 0.236 compared to 0.182 for Scaffolded AI, indicating that

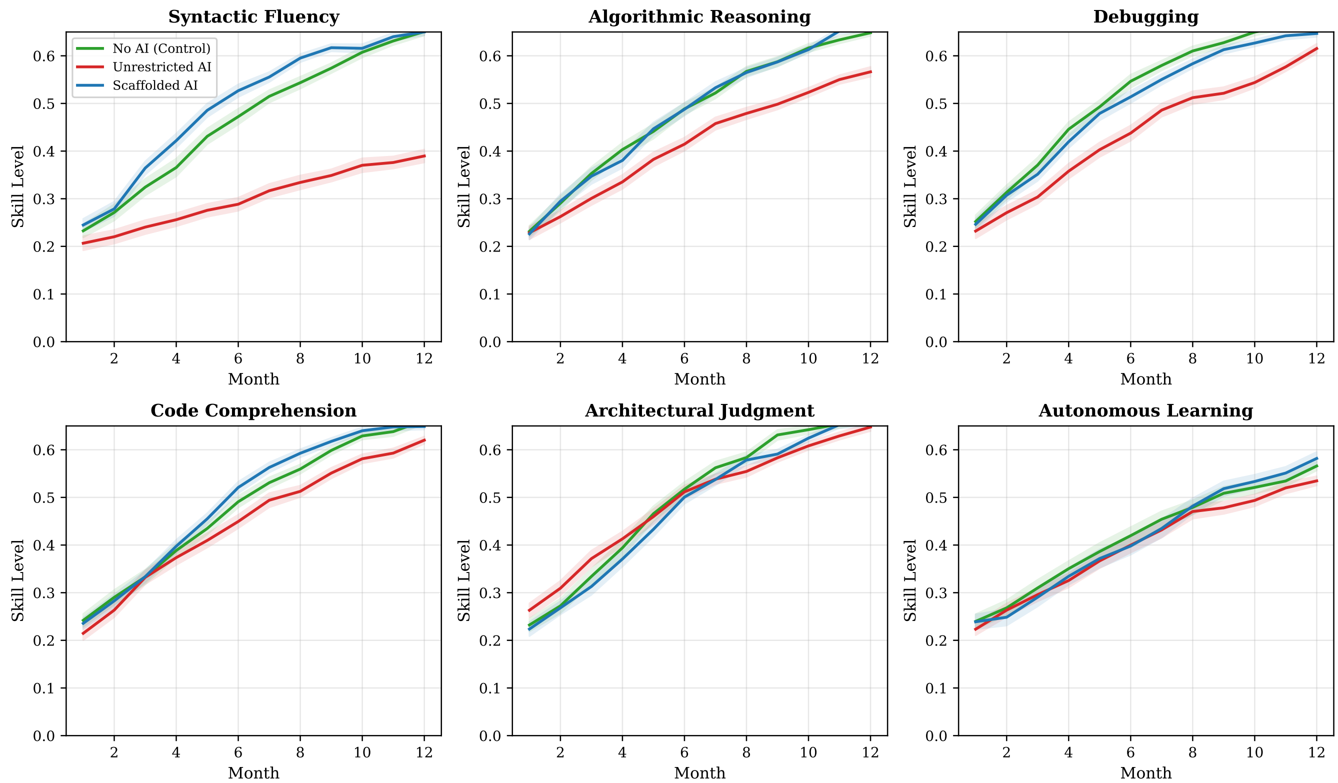

Figure 1: Skill trajectories across six programming dimensions over 12 months. Lines show group means; shaded regions show 95% confidence intervals. The Unrestricted AI condition (red) shows progressively diverging trajectories from Control (green), with the largest gaps in highly automatable dimensions (syntactic fluency, algorithmic reasoning). The Scaffolded AI condition (blue) closely tracks Control across all dimensions.

Table 2: Dimension-specific final skill levels and effect sizes. Cohen's d compares each AI condition against Control; negative values indicate AI-induced skill impairment. p-values from permutation tests (5,000 permutations). Dimensions ordered by AI automation weight (descending).

	AI Weight	Final Skill (Mean)			Cohen's d vs. Control	
Dimension	w_i	Control	Unrest. AI	Scaff. AI	Unrest. (p)	Scaff. (p)
Syntactic Fluency	0.80	0.651	0.390	0.650	−5.10 (< .001)	−0.02 (0.910)
Algorithmic Reasoning	0.50	0.648	0.566	0.660	−2.07 (< .001)	+0.34 (0.031)
Debugging	0.35	0.666	0.615	0.647	−1.28 (< .001)	−0.59 (< .001)
Code Comprehension	0.25	0.662	0.620	0.649	−1.21 (< .001)	−0.42 (0.010)
Architectural Judgment	0.15	0.664	0.648	0.656	−0.44 (0.006)	−0.22 (0.177)
Autonomous Learning	0.10	0.566	0.535	0.582	−0.72 (< .001)	+0.30 (0.065)

unrestricted users remain more dependent on AI tools despite 12 months of practice.

3.5 Sensitivity Analysis: The Crossover Threshold

Figure 5 presents the sensitivity analysis varying processing depth ϕ from 0.05 to 0.95. Below $\phi \approx 0.75$, AI assistance produces a net negative effect on skill formation. Above this threshold, the learning benefit of reduced difficulty and increased success rate outweighs the cost of reduced cognitive effort, and AI becomes net-positive.

This crossover threshold at $\phi = 0.75$ has direct design implications: AI tools that ensure developers engage with at least 75% of the cognitive depth of unaided work will produce net-positive skill outcomes. The default Unrestricted AI processing depth of 0.15 falls far below this threshold, explaining the large negative skill effect. The Scaffolded AI condition's processing depth of 0.70 approaches

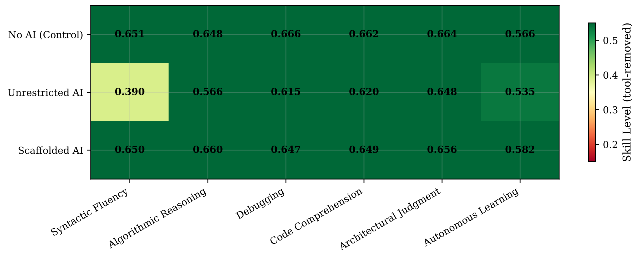

Figure 2: Heatmap of final skill levels by condition and dimension. Warmer colors indicate higher skill. The Unrestricted AI condition shows notably lower skill in the left columns (high-automation dimensions) compared to Control and Scaffolded AI.

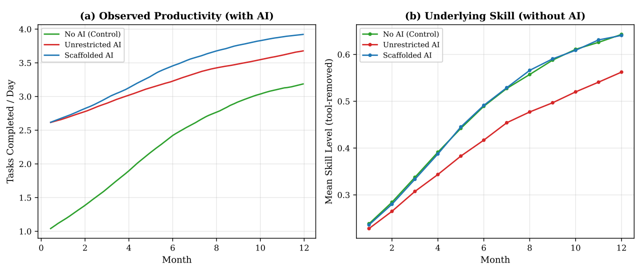

Figure 3: The productivity–skill dissociation. (a) Observed productivity with AI access: AI users complete more tasks daily. (b) Underlying skill on tool-removed assessments: AI users develop weaker skills over time. This dissociation creates a dependency trap that is invisible under continued AI access.

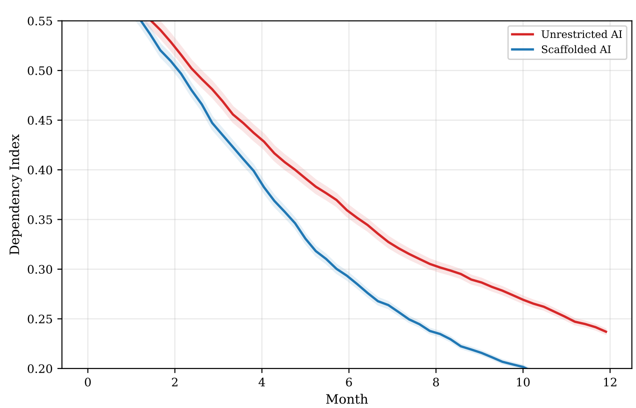

Figure 4: Dependency Index (DI) over 12 months. Higher values indicate greater reliance on AI tools. Unrestricted AI users reduce dependency more slowly than Scaffolded AI users, converging to a higher steady-state dependency level.

but does not quite reach the threshold, explaining its near-neutral overall effect.

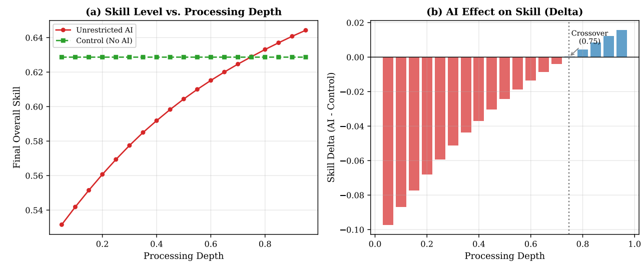

Figure 5: Sensitivity analysis. (a) Final skill levels as a function of cognitive processing depth during AI-assisted work. (b) Skill delta (AI minus Control): the crossover from negative to positive occurs at processing depth ≈ 0.75. Below this threshold, AI harms skill formation; above it, AI helps.

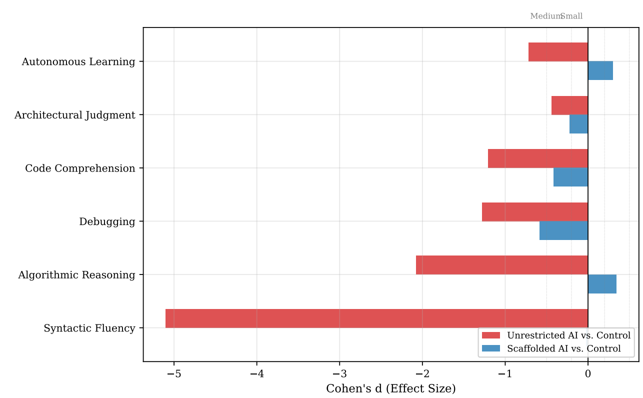

Figure 6: Cohen's d effect sizes by dimension. Unrestricted AI (red) shows consistently negative effects, largest for highly automatable skills. Scaffolded AI (blue) shows near-zero effects across most dimensions, with modest positive effects for algorithmic reasoning and autonomous learning.

3.6 Effect Size Summary

Figure 6 displays Cohen's d effect sizes for all six dimensions under both AI conditions compared to Control. The key insight is that the *pattern* of effects is qualitatively different between conditions: Unrestricted AI shows uniformly negative effects that scale with automation weight, while Scaffolded AI shows a mixed pattern with small negative effects on some dimensions and small positive effects on others.

4 DISCUSSION

4.1 The Skill Formation Paradox

Our model predicts a fundamental tension between short-term productivity and long-term skill development. Unrestricted AI use—the default mode in which most novice developers interact with AI tools—produces a large negative effect on skill formation ($d = -1.04$) while simultaneously boosting observable productivity. This *productivity–skill dissociation* creates a systemic risk: organizations optimizing for measurable output will inadvertently produce developers who cannot function without AI scaffolding.

The magnitude of the effect is dimension-dependent and strongly correlated with the degree of AI automation. Syntactic fluency—the skill most readily automated by current AI tools—shows the largest impairment ($d = -5.10$). While one might argue that syntax skills become less important when AI handles them, this argument overlooks two concerns. First, syntactic fluency is foundational; debugging, code review, and architectural reasoning all require the ability to read and write code fluently. Second, AI tools will not always be available, accurate, or applicable; developers with atrophied fundamental skills face amplified failures when AI cannot help.

4.2 Scaffolding as a Solution

The Scaffolded AI condition demonstrates that the negative skill effect is not inherent to AI tool use but rather to the *mode of engagement*. When novices are required to actively process AI output—reading, modifying, and explaining generated code before incorporating it—skill development proceeds at nearly the same rate as unaided practice ($d = -0.04$). This finding aligns with prior work on active learning and desirable difficulty [3] and suggests concrete design interventions:

- **Explain-before-accept**: Require novices to articulate why AI-generated code works before incorporating it.
- **Modification prompts**: Present AI suggestions in a form that requires adaptation rather than verbatim acceptance.
- **Interleaved practice**: Periodically disable AI assistance to force unscaffolded practice.
- **Progressive withdrawal**: Gradually reduce AI assistance as skill levels increase, analogous to training wheels.

4.3 The Crossover Threshold

The sensitivity analysis identifies a processing depth threshold of $\phi \approx 0.75$ at which AI transitions from skill-harming to skill-enhancing. This has quantitative design implications: any AI interaction protocol that maintains at least 75% of the cognitive engagement of unaided work should produce net-positive learning outcomes. Current AI tools that offer frictionless code completion (estimated $\phi \approx 0.15$) are far below this threshold, while structured engagement protocols can approach or exceed it.

4.4 Limitations

Our findings are based on a computational model, not empirical data from human participants. The model makes assumptions about cognitive architecture (learning rates, forgetting dynamics, transfer structure) that, while grounded in established theory, may not precisely match real-world learning. Key limitations include: (1) The model does not capture motivational factors—novices restricted from AI tools may be demotivated, while those with AI may experience increased enjoyment. (2) The task environment is simplified; real software development involves social interaction, code review, and collaborative problem-solving that may modify learning dynamics. (3) The processing depth parameter, while theoretically motivated, conflates multiple cognitive processes into a single scalar. (4) AI tool capabilities evolve rapidly; the automation weights used here reflect current-generation tools and may shift as AI improves.

These limitations are inherent to the computational modeling approach but are offset by its strengths: the ability to generate precise, testable predictions; systematic exploration of parameter space; and low cost relative to longitudinal human studies.

4.5 Empirical Validation

Our model generates several testable predictions for empirical studies:

1. **Dimension-specificity**: The AI-induced skill deficit should be largest for syntactic and algorithmic skills, smallest for architectural and meta-cognitive skills.
2. **Engagement moderation**: Active engagement protocols should substantially reduce or eliminate the skill deficit.
3. **Dependency trap**: Tool-removed assessments should reveal skill gaps invisible in AI-assisted performance metrics.
4. **Threshold effect**: Interventions increasing processing depth above ~ 0.75 should flip the AI effect from negative to positive.

We recommend a Randomized Longitudinal Skill Assessment (RLSA) design—a 12-month, three-arm trial with monthly tool-removed assessments across all six skill dimensions—as the empirical study most directly suited to testing these predictions.

5 CONCLUSION

We have presented a computational cognitive model that addresses the open question of how AI coding tools affect novice developer skill formation. Our simulation of 240 developers over 12 months reveals a *skill formation paradox*: unrestricted AI use boosts productivity while significantly impeding underlying skill development, with the strongest effects in highly automatable skill dimensions. Critically, scaffolded engagement—requiring active processing of AI output—nearly eliminates this deficit, and sensitivity analysis identifies a processing depth threshold at $\phi \approx 0.75$ that separates skill-harming from skill-enhancing AI use.

These findings have immediate practical implications. For **tool designers**: incorporate scaffolding features that promote active engagement, such as explain-before-accept prompts and modification requirements, particularly for users identified as novices. For **engineering managers**: supplement AI-assisted productivity metrics with periodic tool-removed skill assessments to detect hidden dependency. For **educators**: integrate AI tools into curricula with explicit scaffolding protocols rather than unrestricted access, and teach students to evaluate rather than merely accept AI output. For **researchers**: prioritize empirical studies that disentangle productivity from skill, measure multiple skill dimensions, and test engagement-mode interventions.

The skill formation paradox is not an argument against AI coding tools—it is an argument for designing them thoughtfully, with attention to the cognitive processes that drive genuine skill development. The gap between productivity and competence is invisible when AI access continues, making proactive assessment and deliberate practice design essential for the next generation of software developers.

REFERENCES

[1] John R. Anderson. 1982. Acquisition of Cognitive Skill. *Psychological Review* 89, 4 (1982), 369–406.

[2] Hamsa Bastani, Osbert Bastani, Alp Sungu, Haosen Ge, Ozge Kabakcı, and Rei Mariman. 2024. Generative AI Can Harm Learning. *arXiv preprint arXiv:2410.15745* (2024).

[3] Robert A. Bjork. 1994. Memory and Metamemory Considerations in the Training of Human Beings. In *Metacognition: Knowing About Knowing*. MIT Press, 185–205.

[4] Robert A. Bjork and Elizabeth L. Bjork. 1992. A New Theory of Disuse and an Old Theory of Stimulus Fluctuation. *From Learning Processes to Cognitive Processes: Essays in Honor of William K. Estes* 2 (1992), 35–67.

[5] Zheng Cui, Alejandra Zambrano, Jerry Lo, Michael Lee, and Juho Leinonen. 2024. The Effects of Generative AI on Computing Students' Help-Seeking Preferences. *arXiv preprint arXiv:2410.12944* (2024).

[6] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A. Becker, and Brent N. Reeves. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (2024), 56–67.

[7] Yuxiang Hou, Siddharth Bhatt, Jiawen Zang, Yash Agarwal, Sida Wu, and Sida Peng. 2024. The Effects of Generative AI on High Skilled Work: Evidence from Three Field Experiments with Software Developers. *arXiv preprint arXiv:2410.12944* (2024).

[8] Slava Kalyuga, Paul Ayres, Paul Chandler, and John Sweller. 2003. The Expertise Reversal Effect. *Educational Psychologist* 38, 1 (2003), 23–31.

[9] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (2023), 1–23.

[10] Kenneth R. Koedinger, Albert T. Corbett, and Charles Perfetti. 2012. The Knowledge-Learning-Instruction Framework: Bridging the Science-Practice Chasm to Enhance Robust Student Learning. *Cognitive Science* 36, 5 (2012), 757–798.

[11] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. *arXiv preprint arXiv:2210.14306* (2024).

[12] Raja Parasuraman and Victor Riley. 1997. Humans and Automation: Use, Misuse, Disuse, Abuse. *Human Factors* 39, 2 (1997), 230–253.

[13] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590* (2023).

[14] James Prather, Brett A. Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2024. The Widening Gap: The Effects of AI-Assisted Code Generation on Novice and Expert Developers. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education* (2024), 142–148.

[15] Zheyuan Shen, Nikolas Zolas, Samuel Assefa, Miranda Bogen, and Noam Slonim. 2026. How AI Impacts Skill Formation. *arXiv preprint arXiv:2601.20245* (2026).

[16] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (2022), 1–7.

[17] Celina Weber, Linwei Fang, David Broneske, and Gunter Saake. 2025. The Impact of AI Tools on Software Engineering Tasks. *arXiv preprint arXiv:2507.09089* (2025).