The AI coding slop loop will eat its own gains

Kabui, Charles

2026-07-15

Table of Contents

Introduction 2
1. Training data is already part of the problem 3
2. Generated data can also be useful 3
3. Code arrives faster than review 4
4. Maintenance is where the gain can disappear 4
5. Skills move context, not responsibility 5
6. The five-line loop is not the whole job 5
7. How to stop feeding the loop 6
8. Conclusion 6
References 7

Introduction

AI can write a five-line loop in seconds. That is the easy part.

Suppose the patch works well enough to merge. Six months later, another agent has to change it. The repository now has extra tests, comments, instruction files, wiki pages, and perhaps a skill meant to prevent the first agent's mistake. The second agent must sort through that pile before touching the same five lines. Some of it may eventually turn up in a training set.

This is the **AI coding slop loop**: generated material makes later generation harder, inviting yet more generated material to explain and repair it. Researchers have measured the parts separately. Training on model output can narrow what models learn. Repositories are accumulating copied code. Long prompts make models less reliable. Developers spend time debugging answers that looked right.

Nobody has followed one project through the full cycle long enough to prove it runs away. Model capability can improve while a project's maintenance burden gets worse. The research supports a less dramatic warning: **AI can move work from writing code to reviewing, explaining, and maintaining it until the early speed gain is gone.**

---

1. Training data is already part of the problem

Model makers already train on machine-made material. Microsoft's Phi-4 report says synthetic data makes up the **bulk of its training data**. Microsoft used several methods, including model agents and self-revision Abdin et al. (2024). AI learning from AI is already an engineering choice.

Trouble starts when generated material replaces varied original material. For the Nature paper *"AI Models Collapse When Trained on Recursively Generated Data"*, Shumailov et al. (2024) repeatedly trained new models on output from earlier ones. Rare parts of the original data disappeared first. Later generations gave a worse account of the whole distribution. The loss went beyond a few bad answers: less common possibilities disappeared.

Software has plenty of those rare cases. A routine API call appears everywhere; a workaround for one revision of an old controller may exist in a single repository. If training data fills up with conventional AI output, models see more copies of the routine answer and fewer examples of the strange answer a real machine sometimes needs.

The experiment measures one training setup. It cannot forecast every model release.

---

2. Generated data can also be useful

Calling all synthetic data slop would be a mistake. Microsoft's 14-billion-parameter Phi-4 beat its GPT-4 teacher on several science and maths question sets Abdin et al. (2024). The team designed material for particular skills, checked it, and mixed it with other sources.

Gerstgrasser et al. (2024) tested this distinction in the ICML paper *"Is Model Collapse Inevitable?"*. Test error rose when each generation's output replaced the original dataset. **Keeping the real data and adding generated data avoided collapse** across different model sizes, designs, and training settings.

A trainer needs to know whether a file came from a carefully reviewed implementation or from ten generated copies. A broad scrape that treats them as equivalent throws that information away.

The risky recipe is **unlabelled, repetitive, weakly checked model output that crowds out rare human examples**. Model builders can avoid it. Without that care, a larger dataset may contain less useful variety.

---

3. Code arrives faster than review

The speed gain is real for many people. Google's 2025 DORA report drew on nearly **5,000 technology professionals** and more than 100 interview hours. **90% used AI at work**, and more than 80% believed it made them more productive Harvey and DeBellis (2025).

Faster typing changes what lands in the repository. GitClear analysed **211 million changed lines** from 2020 through 2024. Copied code rose from **8.3% of changed lines in 2021 to 12.3% in 2024**. Meanwhile, the share associated with moving and improving existing code fell from 25% to below 10% GitClear (2025). Its earlier dataset found more code changed or reverted within two weeks.

GitClear cannot identify who or what wrote every line, so the data does not prove that AI caused the shift. Adding a new helper is easy to count. Finding the old one and deleting the duplicate takes more knowledge while producing fewer lines.

A team can merge more code and look faster while its backlog quietly changes. Instead of features waiting to be written, it has software waiting to be understood.

---

4. Maintenance is where the gain can disappear

The 2025 Stack Overflow Developer Survey received more than **49,000 responses**. Of the 31,476 people who answered its question about AI frustrations, **66% had received solutions that were almost right**. Another **45.2% said debugging AI-generated code took more time**, and 16.3% struggled to understand the code Stack Overflow (2025). Self-reports cannot measure time precisely, but cleanup is plainly part of AI-assisted work.

METR tested 16 experienced open-source developers on 246 real issues in repositories they knew well. With early-2025 AI tools, they took **19% longer**, although they expected to become 24% faster Becker et al. (2025). METR now calls that result outdated. In February 2026 it said newer tools probably help more, but developers who depend on AI increasingly refuse no-AI tasks, which makes comparisons harder METR (2026).

Another experiment came out differently. Borg et al. studied **151 participants**, 95% professional developers. People using AI finished a Java task **30.7% faster**. Other developers later changed that code without AI, with no significant penalty in time or code health Borg et al. (2026).

The studies examine different work. A bounded Java task is unlike a mature repository full of unwritten rules. DORA found a similar split: more AI use went with better throughput and product performance, but **worse delivery stability** Harvey and DeBellis (2025). AI can save time initially and send part of the bill downstream.

5. Skills move context, not responsibility

Skills are a sensible response to repeated mistakes: write the rule once and load it only when needed. Anthropic's design tries to contain that cost.

Only a skill's name and description, about **100 tokens per installed skill**, are always present. Full instructions load when triggered and should stay below **5,000 tokens**. References and scripts stay outside the prompt until needed Anthropic (2025). A hundred well-made skills do not require a million-token greeting.

Staged loading solves prompt size, not upkeep. Someone must choose the trigger, write the instructions, settle overlaps, and update the skill. A stale skill is worse than a stale wiki page because the agent may act on it automatically.

A large context window is not a free filing cabinet. Chroma tested **18 models** over 194,480 calls and found that reliability generally fell as inputs grew, even on simple tasks Hong et al. (2025). In one memory test, focused prompts averaged 300 tokens and beat full histories averaging 113,000. Anthropic advises using the **smallest set of useful tokens** that can do the job Anthropic Applied AI (2025).

A useful skill removes repeated explanation. A bad one joins the pile that the next person, or model, must maintain.

6. The five-line loop is not the whole job

The requested code may be five lines, but the agent first has to locate them. It may search the repository, load instructions, retrieve documentation, generate tests, repair a failed attempt, and summarize the change. The diff hides most of that work.

Anthropic says its agents use about **4 times more tokens than ordinary chat**, while multi-agent systems use about **15 times more** Hadfield et al. (2025). Its multi-agent research system beat a single agent by 90.2% on an internal test. Yet Anthropic says most coding tasks are a poor fit because they have fewer independent branches and more shared dependencies.

The cost can creep upward inside one project. A confusing codebase demands more context and attempts. Failures produce new tests, notes, and rules. The next agent has a larger pile to inspect, so the team writes another skill.

I have not found a long-term study showing that the same `for` loop costs more with every model generation. That remains a hypothesis. We do know that long context hurts reliability,

agents spend multiples of ordinary chat tokens, and unstable code moves work downstream. Count that work, not only the five-line diff.

---

7. How to stop feeding the loop

Most teams do not need another AI policy document. They need habits that expose unnecessary code and context.

- Keep verified human data and label generated material. Recursive training research found that retaining original data can prevent collapse Gerstgrasser et al. (2024).
- Measure review time, rework, incidents, deleted code, and duplication. "Lines generated" rewards expansion.
- Put a person's name on every AI-assisted change. That person should explain the design, failure modes, and tests before merge.
- Ask the agent to search for an existing implementation before writing one. Review generated documentation like generated code.
- Keep skills narrow, load them only when needed, and delete stale rules. Fixed procedures often belong in scripts, where only the output needs to enter the prompt Anthropic (2025).
- Strengthen tests and feedback before increasing generation. DORA found that these practices help teams turn AI speed into useful throughput rather than instability Harvey and DeBellis (2025).

The aim is to leave less software for somebody to puzzle through later. The faster a tool can produce a line, the stronger the reason should be for keeping it.

---

8. Conclusion

The slop loop is possible, but nobody has followed one codebase for ten years to prove it inevitable.

What we can see is uncomfortable enough. Generated data can erase rare patterns when it replaces original material. Copied code occupies a larger share of repository changes. Developers repair answers that were close enough to waste time, and long prompts make that repair less reliable. Agents can spend far more tokens while doing it.

The research also shows ways out:

- Checked synthetic data can improve a model when original data stays in the mix.

- In controlled work, AI made developers faster without making the result harder to change.
- A focused skill can save context instead of consuming it.
- Good tests and quick feedback catch costs before they settle into the codebase.

Count the time spent understanding, verifying, changing, and eventually deleting the result. A five-line loop that needs fifty thousand instruction tokens and a week of review is not cheap code, however quickly it appeared.

A team that rewards small diffs, deletion, clear ownership, and easy maintenance can use AI without losing its grip on the project. Reward output volume, and the volume will arrive. The rules, documents, skills, and repair work will follow.

References

1. METR. (2026). *We Are Changing Our Developer Productivity Experiment Design.* Research update.
2. Borg, M., et al. (2026). *Echoes of AI: Investigating the Downstream Effects of AI Assistants on Software Maintainability.* Empirical Software Engineering.
3. Harvey, N. & DeBellis, D. (2025). *2025 DORA Report: State of AI-Assisted Software Development.* Google Cloud DORA.
4. Stack Overflow. (2025). *2025 Developer Survey: AI.* Stack Overflow.
5. Hong, K., Troynikov, A. & Huber, J. (2025). *Context Rot: How Increasing Input Tokens Impacts LLM Performance.* Chroma Technical Report.
6. Becker, J., et al. (2025). *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity.* arXiv:2507.09089.
7. Hadfield, J., et al. (2025). *How We Built Our Multi-Agent Research System.* Anthropic Engineering.
8. Anthropic Applied AI. (2025). *Effective Context Engineering for AI Agents.* Anthropic Engineering.
9. Anthropic. (2025). *Agent Skills Overview.* Claude Developer Platform.
10. GitClear. (2025). *AI Copilot Code Quality: 2025 Look Back at 12 Months of Data.* GitClear Research.
11. Shumailov, I., et al. (2024). *AI Models Collapse When Trained on Recursively Generated Data.* Nature, 631, 755-759.
12. Gerstgrasser, M., et al. (2024). *Is Model Collapse Inevitable? Breaking the Curse of Recursion by Accumulating Real and Synthetic Data.* ICML 2024.
13. Abdin, M. I., et al. (2024). *Phi-4 Technical Report.* Microsoft Research Technical Report MSR-TR-2024-57.

Disclaimer: *For information only. Accuracy or completeness not guaranteed. Illegal use prohibited. Not professional advice or solicitation.* ***Read more:*** */terms-of-service*