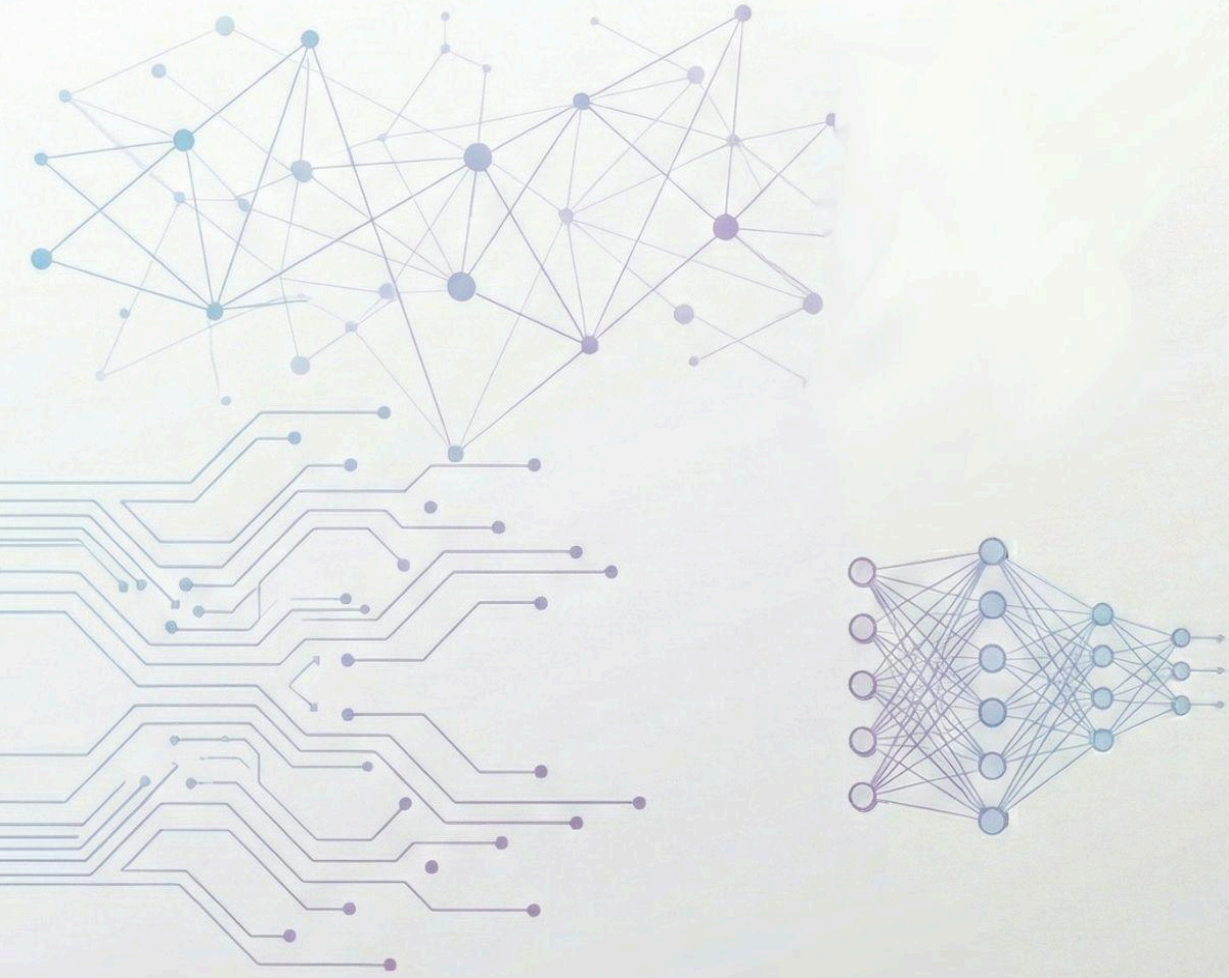
AI Coding Tools Attract Top Performers – But Do They **Create** Them?

Evidence from 2,172 developer-weeks on the connection between AI use, productivity & durable code generated. Featuring cohort data retrieved from Cursor, Github Copilot & Claude Code.



William Harding, Lead Developer & CEO
GitClear Research, in partnership with GitKraken Insights
Published January 4, 2026

GitClear GitKraken

Abstract: High-Fidelity Measurement Reveals Significant Productivity Gap Associated with AI Use
The "Steroid" Effect
Where's the Meat? In Every Productivity Metric Tested
Correlation Traps: What Confounds the AI/Productivity Relation, and What Remains as AI Productivity Gain?
Tallying Confounders: How AI Power Users Are Different
1. Senior and Staff Engineers Often Early AI Adopters
2. Startup Teams Move Fast and Try New Tools First
3. Developers in Flow Use AI More
Beyond Confounders: Meaningful AI Benefits
1. Volume of Test Code Authored is Higher
2. Proportionally Lower Review Time Induced
Quantifying Negative AI Impact
The Problem of Code Churn Lines (Refactoring)
The Problem of Duplication and Copy/Paste
Conclusions
AI Is a Force Multiplier for Developers Who Are Already High-Agency
Developers who already behave like 10x engineers use AI to push even harder.
The Future: AI Use as a Developer Differentiator?
Stories are Fine, Data is Golden
Appendix
A1. Accessing AI Cohort Data
A2. Cohort Data Tables & Year-Over-Year Methodology
A3. GitClear's History as a Source of AI Data
A4. Copy/Paste Anecdata
A5. Credits
A6. Version History
References

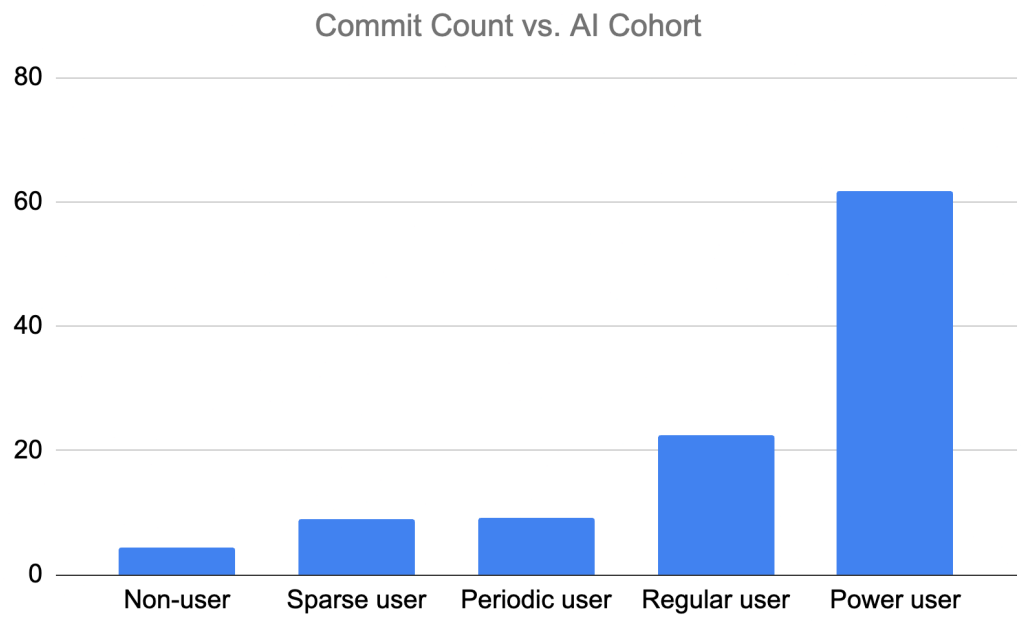
Abstract: High-Fidelity Measurement Reveals Significant Productivity Gap Associated with AI Use

tl;dr AI tools correlate with 4-10x output differences, but confounding factors (seniority, company stage, flow state) explain most of this gap. Actual year-over-year improvements are ~25%, with significant increases in test coverage but 9x higher code churn.

The first quarter of data from GitClear's AI usage measurement is in, and it defies conventional wisdom. Developers who use AI throughout the day aren't just 10% faster—empirical data shows them authoring 4x more work than "AI non-users" during the weeks their AI use is highest [A1].

However, our research suggests this isn't another story about the triumph of AI tools and agents. Moreso, this story is about ***who is using the tools*** and ***what company policies must prospective AI developers contend with***? It's also a story about the risks that come with accelerated code generation: these AI-infused developers churn code at a 9x higher rate.

Constructed from AI provider APIs, the new data lets us compare developers whose weekly AI use ranged from "Non-User" to "Power User" [A1]. Contrary to our expectations, the difference in productivity data between the two sides amounts to a chasm:



Weekly "Commit Count" is far from an end-all productivity metric – but it mirrors the trend found across other proxies for developer output

Finding such an extreme gap implies the existence of something akin to "dark matter" for developer productivity. There must be *much* more than "enlightened AI use" to explain a difference of this magnitude. In the sections to follow, we'll present and dissect data to

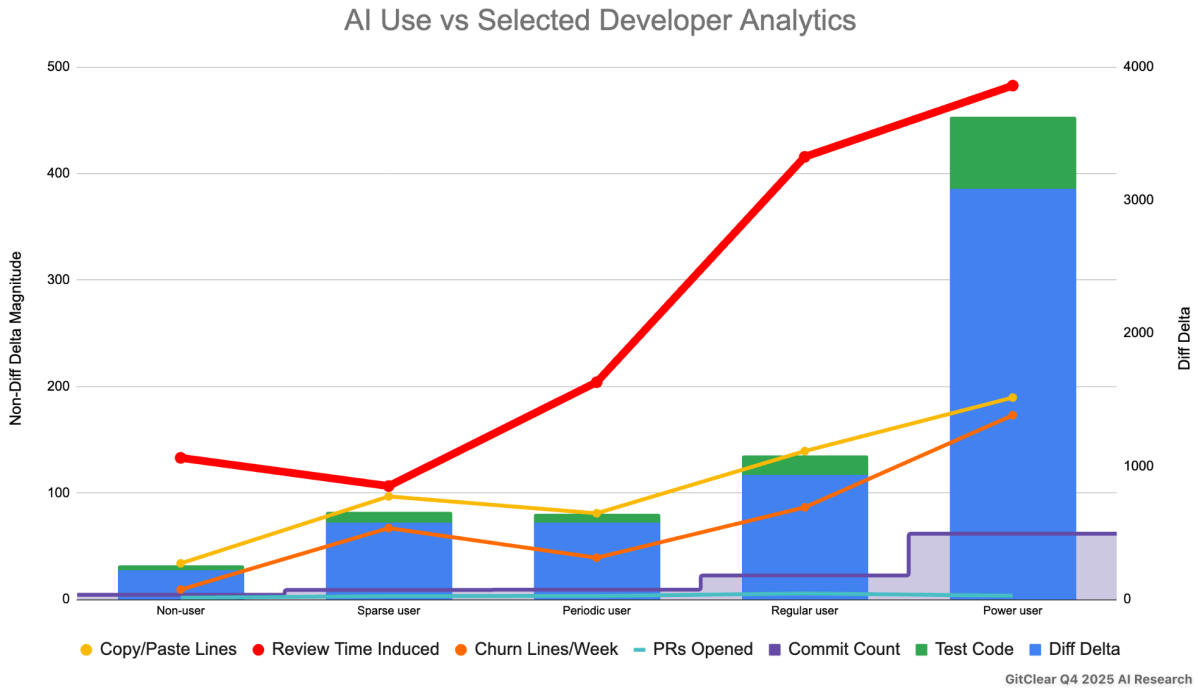
understand “how can the ‘highest engagement’ cohorts average 5x more progress across almost every metric GitClear tested?”

In other words, what’s the source of the “dark matter productivity” enjoyed by the “Power User” cohort? If factored out, how large of a difference remains? That calculation can help executives hone their intuition for “what quantifiable improvement can we expect to observe, if our team succeeds at maximizing AI benefits while minimizing AI side effects?” As past AI research has demonstrated [R1], the latter objective requires careful attention to combatting the **code churn** and **code block duplication** that accompanies contemporary AI use?

The data from this research has become available thanks to GitClear’s 2025 API integrations with Github Copilot, Cursor, and Claude Code [R2] that directly measure AI use **per team** and **developer,** where permitted by API provider. If you run a development team, the GitClear & GitKraken teams can help generate any of the data from this report for your own team. Find more details in the Appendix.

The "Steroid" Effect

Comparing the weekly output of 2,172 developer-weeks with and without measurable AI engagement, the magnitude of effect is staggering. The difference between a developer who doesn't touch AI vs a power user (targeting the top 5 percent of AI users [R3]) is not a “margin of error”-type difference – it is an “orders of magnitude”-type difference.



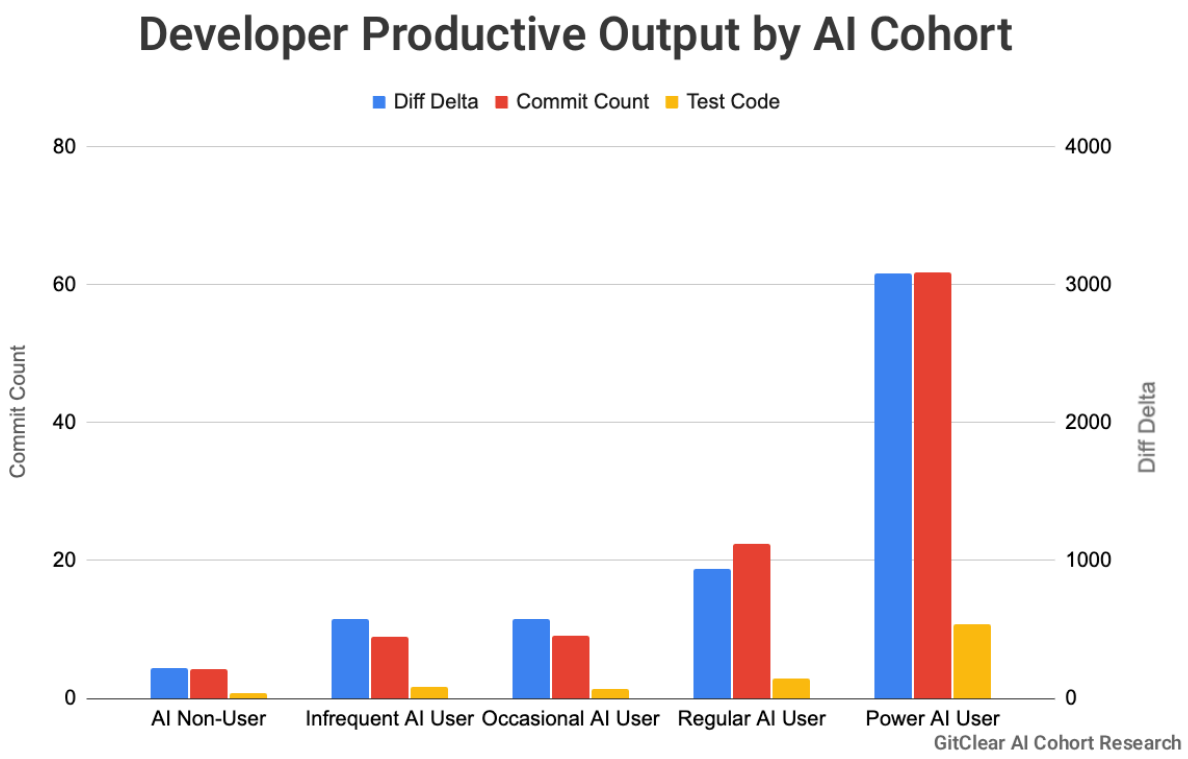
Full data available in Appendix A1

This is the kind of effect size that the LLM providers will likely seize upon as they derive the contours of their marketing push in 2026 and beyond. Yet, as deeper analysis will show, the source of the power user's strength is only marginally the AI tooling itself.

We compare the "power user advantage" to "steroids" because, like steroids in athletics, the performance advantage is real, but comes from a complex interaction of the tool, the user's existing capabilities, and their environment -- not just the tool itself.

Where's the Meat? In Every Productivity Metric Tested

Regardless of which metric is applied to estimate "developer output," substantial differences exist between AI non-users and AI power users:



Across each productivity metric examined, "increased AI use" pairs with "increased output" [A1]

The "Diff Delta" metric — a measure of how much durable, substantive code change is occurring in a repo [R4] — jumps from **221** for non-users to **938** for regular users (3,084 for power users). Commit counts follow a similar trajectory, exploding from **4.3** per week to **22.5** per week for regular users (61 per week for power users). AI-assisted developers are literally implementing code at a rate that exceeds what is possible from a human-speed typist.

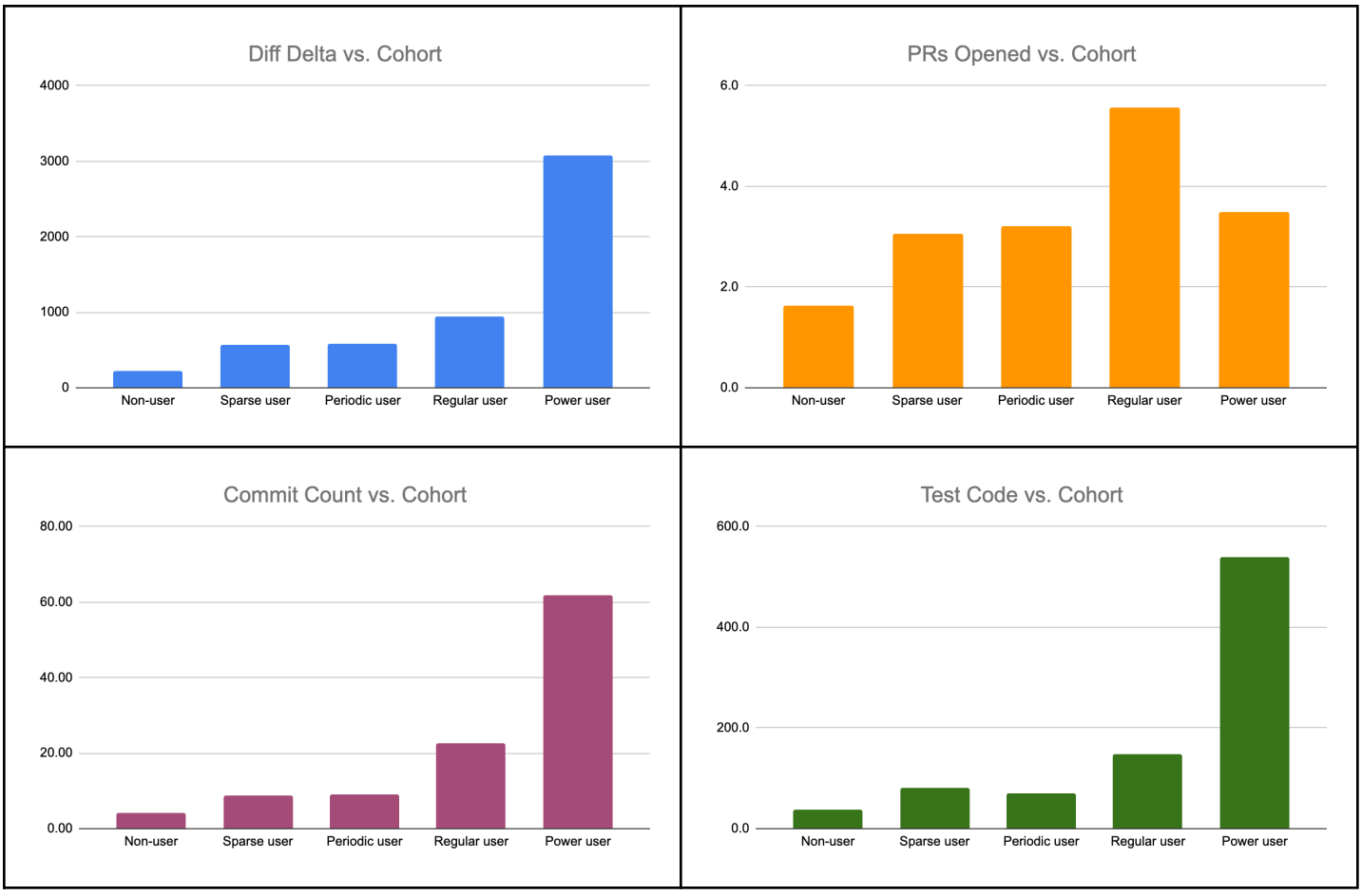
The data suggests this isn't just generating a mountain of burdensome lines to maintain. The "Test Code" metric assesses how much durable code change is being made within test files (unit, integration, system, etc). It shows that the Regular User cohort is generating **147 Diff**

Delta of test code weekly, compared to the Non-User's **37 Diff Delta** in an average week. Power Users are averaging 540 Diff Delta of Test Code. It appears that, when the friction of writing boilerplate tests is removed, developers are more willing to attend to test coverage (a theory validated by survey data from Jetbrains research [R5]). No doubt this tendency is reinforced by the capacity of modern AI agents to iteratively run and re-run a test suite until it passes.

Correlation Traps: What Confounds the AI/Productivity Relation, and What Remains as AI Productivity Gain?

When GitClear began processing data from Github Copilot, Cursor and Claude Code, there were no expectations about whether “the most engaged AI users” would be the most prolific code authors. The conventional wisdom of 2025 seemed to be that AI might increase meaningful output by something like 5-20% (“meaningful output” is much different than “added lines,” which our prior research showed was strongly increasing as AI proliferated [R1]).

When the measured data emerged from its processing method, the trendline was so clear it forced us to double- and triple-check the implementation of the method. Real world data is rarely so unambiguous:

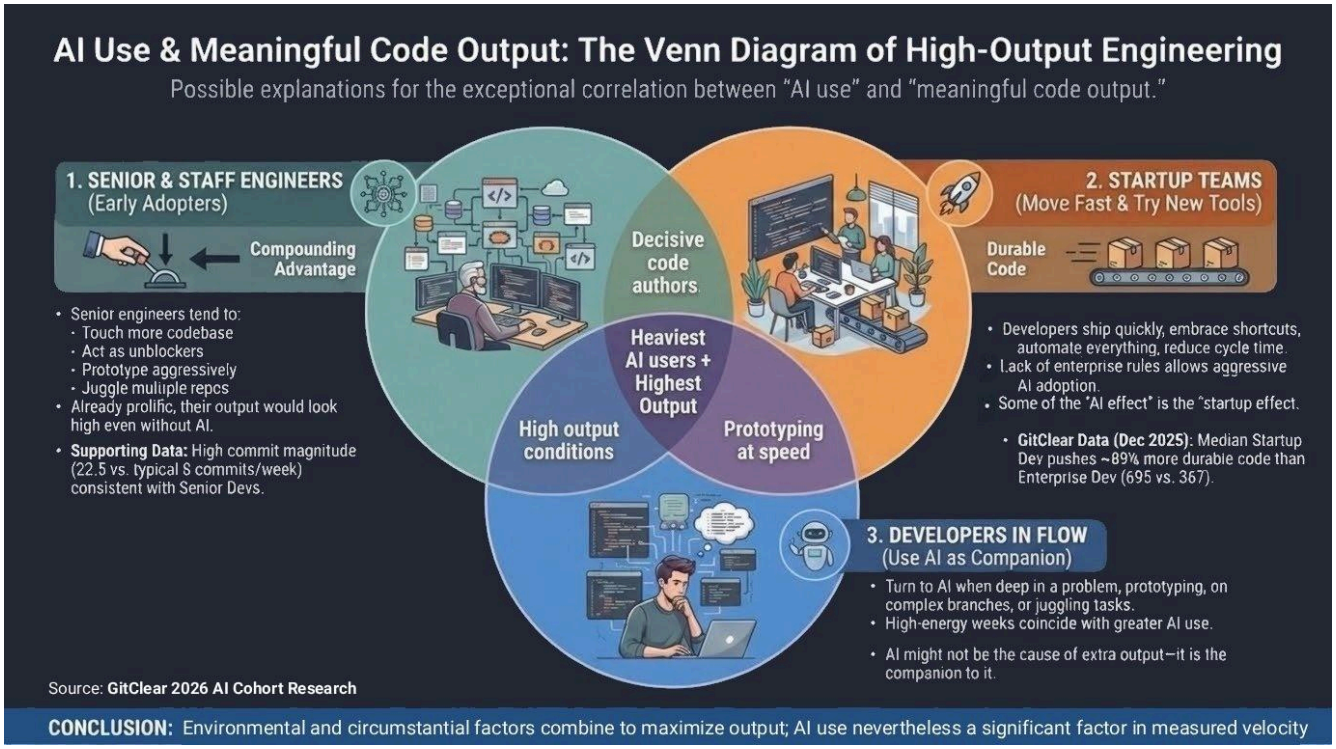


These graphs implausibly assert that “Regular” and “Power” AI Developers generate 4-10x more durable code than Non-Users. Not only that, but the regression curve from “less use” to “more use” [R3] is consistent, with the exception of Power Users’ “opened pull requests.” Finding no errors in the evaluation method forced us to accept that this output gap is “real,” at least in the sense that “more AI use” correlates strongly with “more durable code change.”

But no credible prior research claims an “AI use” effect size approaching 100%. Even Sundar Pichai estimated a meager 10% improvement boost from Google’s AI use as of June 2025 [R6]. After shaking out the confounders, what quantifiable differences remain? These questions are explored in turn in the next two sections.

Tallying Confounders: How AI Power Users Are Different

Let’s consider three explanations that can help explain the heavy correlation between “AI use” and “meaningful code output.”



Contribution of confounders to the correlation between “high output” and “heavy AI use”

1. Senior and Staff Engineers Often Early AI Adopters

Senior engineers tend to:

- Touch more of the codebase
- Prototype aggressively

- Push code with less third-party review
- Lead initiatives that experiment with tools that may later be adopted by full team

Supporting the theory that Senior/Staff engineers contribute to the observed velocity: the magnitude of commits per week (**22.5 commits/week** for "Regular AI User") is almost 3x a typical contributor's **8 commits/week** [R7]. These Commit Counts are most consistent with Senior Developers.

2. Startup Teams Move Fast *and* Try New Tools First

Developers inside early-stage companies are notorious for methodologies that include:

- Ship fast, write tests later
- Embrace prototyping & shortcuts
- Have less legacy code that needs to squeeze into AI context window
- Be first to try any new idea that can gain an edge vs enterprise competitors

For the reasons above, and for the lack of countervailing forces (e.g., rules about "which tools may be used" at enterprise companies), startups are most likely to adopt AI aggressively. Some of the "AI effect" is surely the "startup effect." GitClear's industry stats show that the average Startup Developer pushes roughly 2x more durable code than the average Enterprise Developer [R4]

Benchmark Diff Delta Percentiles

This table collects all committer-weeks from the past year at select benchmark companies. The name-brand companies featured represent open source contributions made to company-specific projects (e.g., Microsoft VS Code, Facebook React) and evaluated by GitClear using freely available commit data. These measurements are extracted from a fraction of the company's full developer staff. They do not represent a full picture of company-wide performance.

Industry benchmark	10th percentile weekly Diff Delta (per day)	Median weekly Diff Delta (per day)	90th percentile weekly Diff Delta (per day)
Google Developer	167 Δ (33 Δ)	**272 Δ** (54 Δ)	738 Δ (148 Δ)
Apple Developer	182 Δ (36 Δ)	**365 Δ** (73 Δ)	1,175 Δ (235 Δ)
Enterprise Company Developer	132 Δ (26 Δ)	**367 Δ** (73 Δ)	1,725 Δ (345 Δ)
Facebook Developer	192 Δ (38 Δ)	**466 Δ** (93 Δ)	1,374 Δ (275 Δ)
Mid-sized Company Developer	159 Δ (32 Δ)	**546 Δ** (109 Δ)	2,026 Δ (405 Δ)
Microsoft Developer	189 Δ (38 Δ)	**550 Δ** (110 Δ)	1,971 Δ (394 Δ)
Startup-sized Company Developer	171 Δ (34 Δ)	**695 Δ** (139 Δ)	2,734 Δ (547 Δ)

As of December 2025, the median Startup Developer pushes 695/367 = 89% more durable code than the median Enterprise Developer

3. Developers in Flow Use AI More

Developers are more likely to turn to AI when they're:

- Prototyping new features
- Making sweeping upgrades to legacy systems
- Juggling multiple tasks, or contemplating multiple approaches
- Unencumbered by management or meetings

High-energy programming weeks coincide with greater AI use. So AI might not be the *cause* of extra output—it might be the *companion* to it.

Even with these confounders, the data retains a clear directionality: **AI use clusters tightly with high-output engineering behavior.** Having identified major confounders, we can now turn to examine what AI-specific benefits remain after accounting for these factors.

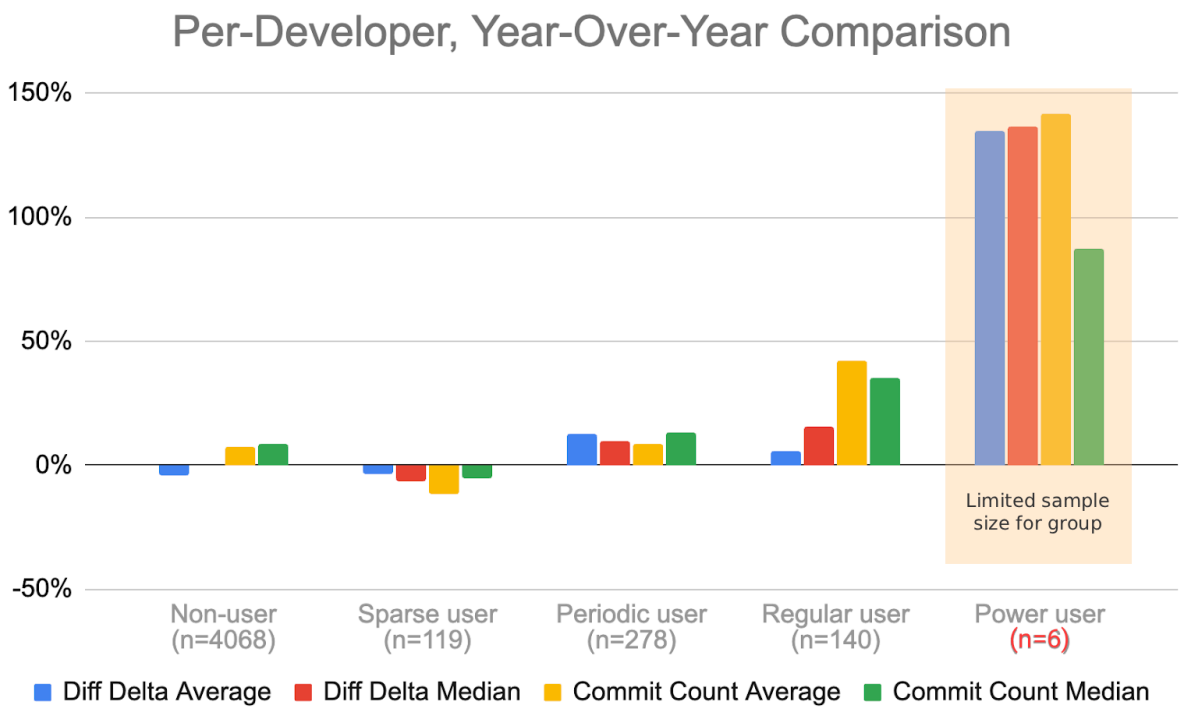
Beyond Confounders: Meaningful AI Benefits

The first follow-up question suggested by collaborators who reviewed our initial findings:

To what extent were high-performing AI users already thriving a year prior, when AI agents were less pervasive, and AI tools more primitive?

If AI is not only **attracting** top performers but **creating** them, we would expect to see a clear uptick in developer output when comparing the year-over-year data of a top developer vs. their past self, evaluated 12 months earlier.

Here is the year-over-year per-developer comparison, evaluating what got done in 2025 vs the same week in 2024, per cohort level:



"Periodic," "Regular," and "Power" AI groups show improvement in their output compared to their past year's performance, although "Power" group size too small for statistical significance. Each bar corresponds to the YoY difference in the stat, comparing a week in 2025 to the same week during 2024.

These numbers fall much closer to existing estimates of “AI Benefit,” compared to the 4-10x output multiples seen in the raw AI cohort comparison data.

Year-over-year data supports the hypothesis that AI has offered the most pronounced benefits to those who were already top performers. Averaging the “Median Commit Count” and “Median Diff Delta” to create a “Hybrid Output” score, allows comparing “Regular User” to “Non-User,” which finds a **4.2x difference** in Hybrid Output. Whereas, comparing “Regular User” to *themselves*, 12 months earlier, yields a comparatively modest 25% increase in output.

Other correlated developer benefits that deserve consideration:

1. Volume of Test Code Authored is Higher

Comparing the volume of test code authored between the cohorts:

Non-Users: **37 test code Diff Delta**
Regular Users: **147 test code Diff Delta** (4.0x higher vs Non-User)
Power Users: **540 test code Diff Delta** (14.6x higher vs Non-User, small sample)

Regular Users show a **4x increase** in the volume of test code written vs non-users. Power Users (with their limited sample size) show a 14x increase. Though AI is causing refactoring to occur at a significantly higher rate (see “Quantifying Negative AI Impacts”), it is simultaneously lowering the cost of writing tests to catch the problems created by rapid output.

2. Proportionally Lower Review Time Induced

While it’s true that the higher-tier AI Cohorts induced more code review time than the non-users, the multiple was actually less than the increase in their output: 3.1x more code review time, vs 4.7x more output averaged commits + Diff Delta.

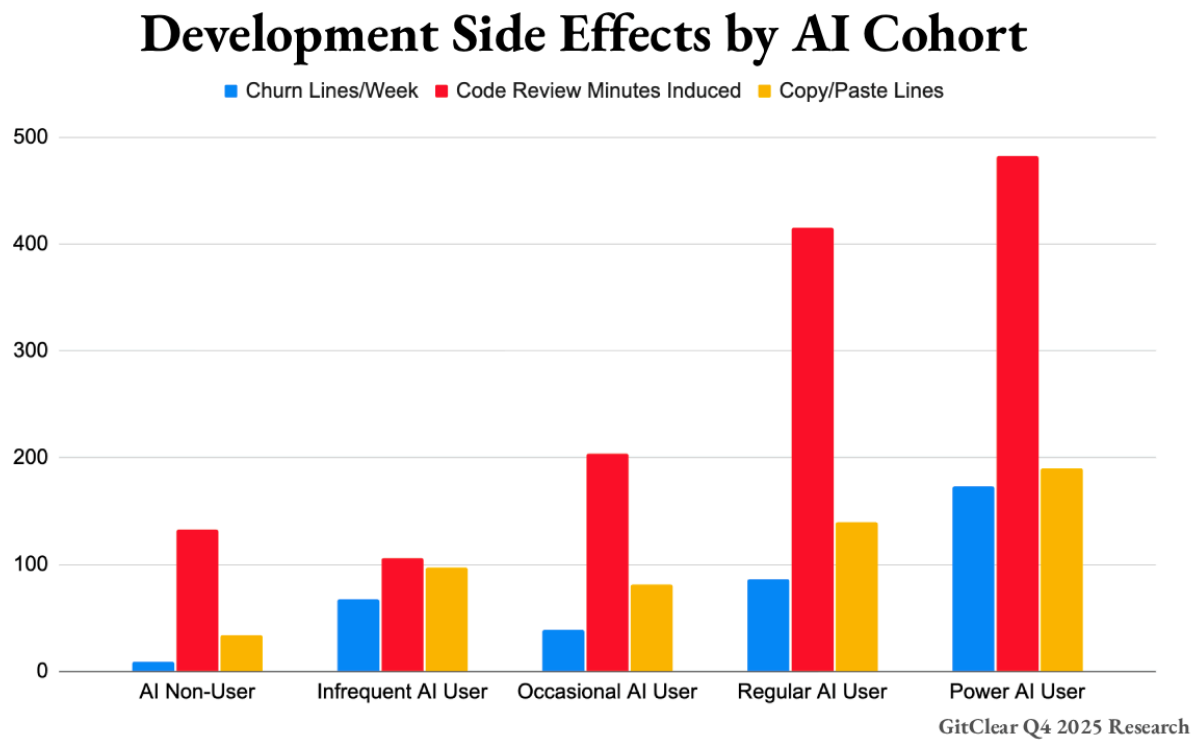
Non-Users: **133 minutes**
Regular Users: **416 minutes** (3.1x higher vs Non-User)
Power Users: **482 minutes** (3.6x higher vs Non-User, small sample)

One factor contributing to “lower review time” is the dip in “PRs Opened” when comparing the “Power” to “Regular” AI user. Likely a combination of confounders #1 (Senior & Staff developer bias) and #2 (Startup-bias) mentioned above. Still, relative to AI skeptics who have supposed that AI is prone to generate low-quality boilerplate, this result suggests that Senior Developers are polishing that boilerplate (via heavy churn) into sufficiently high-quality code by the time it reaches the pull request review stage.

Whether this dip in “code review” leads to a higher defect rate is a question GitClear intends to explore in future research.

Quantifying Negative AI Impact

It should come as little surprise that the most prolific code authors produce a greater net volume of side effects from their changes. “Regular” and “Power” users generate **greater code churn**, **code duplication**, and **code review time from teammates**:



While an increase is apparent in "minutes required to review a developer's code" (**3.1x higher** among Regular AI users vs non-users) and "copy/paste duplication" (**4.1x higher** in Regular AI users vs non-users), that increase is equal or less than the increase in developer output (**4.2x higher**, as detailed in previous section).

The two most concerning trends from heavy AI use are “code churn” and “propensity for code duplication.”

The Problem of Code Churn Lines (Refactoring)

Non-users: **9 lines/week**
Regular users: **87 lines/week**
Power users: **173 lines/week**

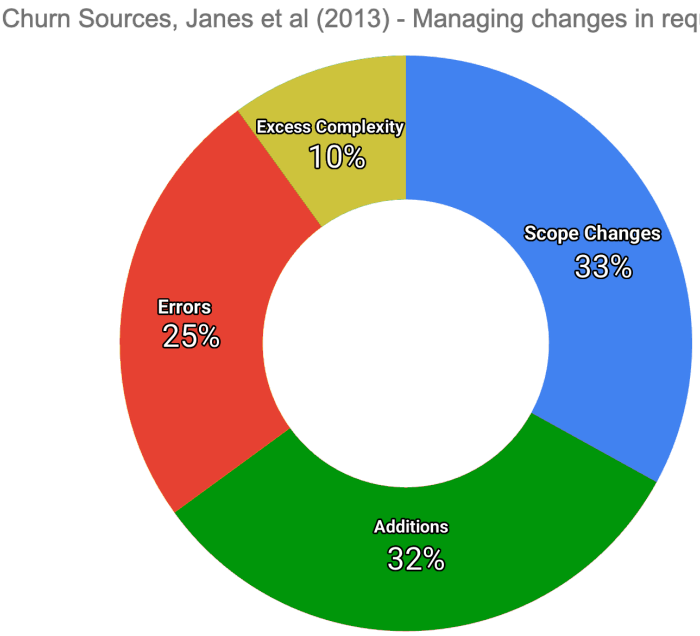
This is the single largest difference found between the opposite ends of the “AI use” spectrum. Relative code churn far outpaces increases in productive output. Regular AI users averaged

9.4x higher code churn than their non-AI counterparts, which is **2.2x greater** than their composite productivity increase:

Metric	**AI Multiple**	**Productivity Multiple** [1]	**Percent**
Code Review Minutes	3.1	4.2	74%
Copy/Paste Lines	4.1	4.2	98%
Churn Lines	9.4	4.2	224%

[1] Difference between "Regular User" and "Non-User," averaged across "Diff Delta," "Commit Count," "PRs Opened," and "Test Delta"

"Churn Lines/Week" represents code that is written and then almost immediately rewritten or deleted. Over-simplified articles sometimes characterize code churn as "flailing about," but, according to Janes et al [R8], churn originates from a variety of causes:



Churned code has many causes; these causes accelerate with heavy AI use

Which is to say, the metric is far from an “unalloyed signal of bad code.” However, it does predict challenges to generating consistent, reliable code: When circumstances arise that induce high churn (e.g., indecisive product direction), the onus is on the developer to demonstrate the conscientiousness to revisit past work.

What makes "code churn" uniquely dangerous for AI users? That it requires an unprecedented level of engineering discipline to evolve AI suggestions from "barebones functional" to "functional, tested, and maintainable."

Whatever the cause of the churn surge, it is clearly the largest AI risk observed amid the current data set. If AI use doesn't actively *induce* churn, it at least *accompanies* it.

The Problem of Duplication and Copy/Paste

Good news first: While the Q4 2025 data shows that heavier AI use begets 4.1x higher copy/paste volume (in line with trends GitClear has previously reported [R1]), that reduction in quality is (slightly) less than the increase in durable code output reported above. So, thus far, the AI power users are generating Copy/Paste lines at a rate proportional to the increase in their total durable code authored. Given AI's propensity to repeat code, should we expect more?

The absence of skyrocketing Copy/Paste follows from the previous thesis: if "Senior Developers are using AI most vigorously as of late 2025," it makes sense that the most experienced developers would be hyper-aware of the maintenance headaches from duplicated code, and therefore hyper-vigilant to expunge it before reaching production.

Still, given the intrinsic tendency of today's LLMs to miss DRY opportunities, we propose a testable thesis for 2027. By the end of 2026, we predict a bell-curve of "Code Duplication per AI Cohort" will emerge. Such a result would correspond with the natural tendency for "Junior" and "Intermediate" developers (whose AI use should soar in 2026) to be less successful at recognizing newly created code duplication, relative to the Senior Devs that inhabit the top AI usage tiers in this research.

Conclusions

What are the takeaways for teams that want to maximize the ROI from their AI investment?

AI Is a Force Multiplier for Developers Who Are Already High-Agency

The nature of the data analyzed suggests that AI doesn't magically turn every developer into a 10x engineer**.** Rather:

Developers who already behave like 10x engineers use AI to push even harder.

They write more tests because they care about quality while moving fast.
They open more PRs because they're comfortable iterating quickly.
They refactor more because AI lowers the cost of big changes.

AI amplifies their habits.

The best developers adopt workflow accelerators early, and becoming proficient in the use of new “process accelerators” widens the gap between “top performers” and “everyone else.”

The Future: AI Use as a Developer Differentiator?

The data shows a clear productivity gap emerging between developer camps:

- Developers who use AI as a daily extension of their workflow
- Developers who treat AI as a last-resort search engine

The former category is already shipping **4–10x more** across key engineering activities. That’s a huge difference, even if much of it can be ascribed to circumstantial factors.

Counter to anti-AI narratives, nothing in this data suggests that AI makes engineers lazier or more careless. The opposite appears true: higher AI use coincides with **more tests**, **more PRs**, **more refactoring**, and, surprisingly, **proportionally fewer review minutes from teammates**.

On balance, this data looks less like “AI replaces developers” and more like “AI is becoming table stakes for high-output engineering work.” If true, the biggest risk developers face isn’t that AI will take their job. It’s that developers who don’t learn to wield AI will simply be outpaced by those who do.

Stories are Fine, Data is Golden

There are many questions we can pursue for our follow-up research – GitClear releases research on a roughly quarterly cadence. Which of these topics would you most enjoy seeing covered in our next paper?

1. How does churn correlate with defect rates? Google DORA’s work has found that 25% higher AI use translates to roughly 15% higher defect rates, but what can we explicitly measure about the relationship between “high churn developers” and “percent of work spent bugfixing”?
2. What are the methods used by the “Power User” AI Cohort? With their stats falling in “outlier” territory across so many metrics, what can we learn from interviewing those at the leading edge of AI productivity?
3. How does this data change if we segment by LLM or provider?
4. What are the maintenance characteristics of the code being written by AI-powered developers? Does it take longer for developers to change it afterward?
5. How does this data change if we explicitly segment it by the experience level of the developer? Do Junior developers armed with AI create defects at a significantly different rate than Senior developers using the same tools?

Find our contact information in A1, below.

Appendix

Data tables, methodology notes and other relevant considerations.

A1. Accessing AI Cohort Data

Discounts are available in Q1 2026 for teams that help GitClear delve further into measuring the full impact of AI use on startup, mid-sized, and enterprise teams. Email hello@gitclear.com if you are open to participating.

A2. Cohort Data Tables & Year-Over-Year Methodology

All developers sampled are from Commercial teams, ranging from “Startup” to “Enterprise”-sized, for data spanning from September 1, 2025 through December 31, 2025.

Read more about the methodology and data collection used in our companion documentation to the research.

A3. GitClear’s History as a Source of AI Data

Since most AI research is funded by parties that have strong incentives to paint a “rosy” or “grim” story about the influence of AI, we believe it essential to relate the past track record of GitClear as an “AI data source.” Many writers in this space have been frustratingly committed to following a single track that “affirms” or “discredits” AI progress. Existing AI research has often been funded by parties with strong priors, either toward skepticism or enthusiasm. Our goal is measurement without predetermined conclusions.

As a source of AI data, GitClear is best known for its 2024 Coding on Copilot research, and the follow-up, 2025’s AI Code Quality Report. Across both research papers, GitClear has focused on “how much more ‘duplication’ and ‘churn’ are correlated with the rise of AI?” To the extent one could predict GitClear’s bias, our findings have been more aligned with “skeptics” than with “boosters.”

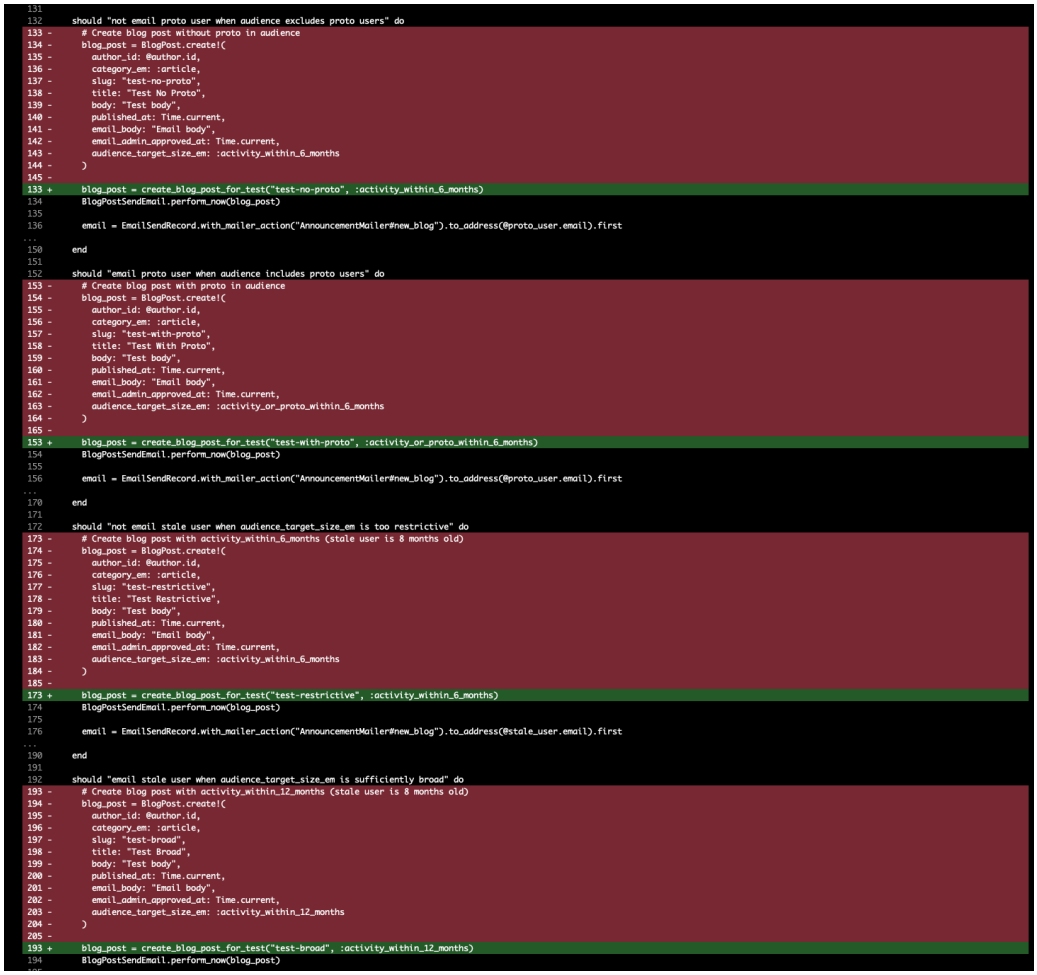
But all of the data we have presented to date has analyzed long-term correlations between the rise of AI, and quantifiable measurements GitClear can extract from this history.

The data available in this report breaks with our prior research in important ways. First, it is *directly measured* data – not just long-term correlation. Second, it finds significant upside associated with greater AI use, albeit with risks to be measured and mitigated to access the full benefit of “rapidly authored, high-quality code output.”

Our primary incentive is to advocate for continued measurement of the impact that AI is having on software development. When measurement reveals substantive benefits, we are determined to report those benefits with the same vigor for data-backed inquiry.

A4. Copy/Paste Anecdata

This example does not constitute any hard evidence, but as a developer that experiments with different AI agents throughout the day, I recently had to request that my LLM revise the test that I originally prompted it to create:



```
131
132     should "not email proto user when audience excludes proto users" do
133 -     # Create blog post without proto in audience
134 -     blog_post = BlogPost.create!(
135 -       author_id: @author.id,
136 -       category_em: :article,
137 -       slug: "test-no-proto",
138 -       title: "Test No Proto",
139 -       body: "Test body",
140 -       published_at: Time.current,
141 -       email_body: "Email body",
142 -       email_admin_approved_at: Time.current,
143 -       audience_target_size_em: :activity_within_6_months
144 -     )
145 -
133 +     blog_post = create_blog_post_for_test("test-no-proto", :activity_within_6_months)
134       BlogPostSendEmail.perform_now(blog_post)
135
136       email = EmailSendRecord.with_mailer_action("AnnouncementMailer#new_blog").to_address(@proto_user.email).first
...
150     end
151
152     should "email proto user when audience includes proto users" do
153 -     # Create blog post with proto in audience
154 -     blog_post = BlogPost.create!(
155 -       author_id: @author.id,
156 -       category_em: :article,
157 -       slug: "test-with-proto",
158 -       title: "Test With Proto",
159 -       body: "Test body",
160 -       published_at: Time.current,
161 -       email_body: "Email body",
162 -       email_admin_approved_at: Time.current,
163 -       audience_target_size_em: :activity_or_proto_within_6_months
164 -     )
165 -
153 +     blog_post = create_blog_post_for_test("test-with-proto", :activity_or_proto_within_6_months)
154       BlogPostSendEmail.perform_now(blog_post)
155
156       email = EmailSendRecord.with_mailer_action("AnnouncementMailer#new_blog").to_address(@proto_user.email).first
...
170     end
171
172     should "not email stale user when audience_target_size_em is too restrictive" do
173 -     # Create blog post with activity_within_6_months (stale user is 8 months old)
174 -     blog_post = BlogPost.create!(
175 -       author_id: @author.id,
176 -       category_em: :article,
177 -       slug: "test-restrictive",
178 -       title: "Test Restrictive",
179 -       body: "Test body",
180 -       published_at: Time.current,
181 -       email_body: "Email body",
182 -       email_admin_approved_at: Time.current,
183 -       audience_target_size_em: :activity_within_6_months
184 -     )
185 -
173 +     blog_post = create_blog_post_for_test("test-restrictive", :activity_within_6_months)
174       BlogPostSendEmail.perform_now(blog_post)
175
176       email = EmailSendRecord.with_mailer_action("AnnouncementMailer#new_blog").to_address(@stale_user.email).first
...
190     end
191
192     should "email stale user when audience_target_size_em is sufficiently broad" do
193 -     # Create blog post with activity_within_12_months (stale user is 8 months old)
194 -     blog_post = BlogPost.create!(
195 -       author_id: @author.id,
196 -       category_em: :article,
197 -       slug: "test-broad",
198 -       title: "Test Broad",
199 -       body: "Test body",
200 -       published_at: Time.current,
201 -       email_body: "Email body",
202 -       email_admin_approved_at: Time.current,
203 -       audience_target_size_em: :activity_within_12_months
204 -     )
205 -
193 +     blog_post = create_blog_post_for_test("test-broad", :activity_within_12_months)
194       BlogPostSendEmail.perform_now(blog_post)
```

If you are a developer that uses AI tools daily, you, too, have surely encountered scenarios like this when you're insufficiently precise in your prompt. The phenomenon wherein LLMs have not generally internalized the heuristic that "less code is better" explains GitClear's research from the past two years, and explains why it seems reasonable to suspect that a lack of increased Copy/Paste can only be explained by an offsetting increase in diligence. The reputation of Staff Engineers is that they are the most conscientious and meticulous developers, which is what earns them the trust to contribute across a broad swath of repos.

A5. Credits

Thank you to Addy Osmani and Nathen Harvey at Google for reviewing earlier drafts of this and suggesting numerous worthwhile improvements. To anyone who was glad the research started with a tl;dr, you have Addy to thank for this time-saving idea. 🙏

A6. Version History

January 7, 2026. Incorporate additional peer review comments & shorten some sections for brevity.
January 4, 2026. Incorporated comments from peer reviewers. Flesh out the “Drawbacks” section.
December 24, 2025. Beta version uploaded to S3.

References

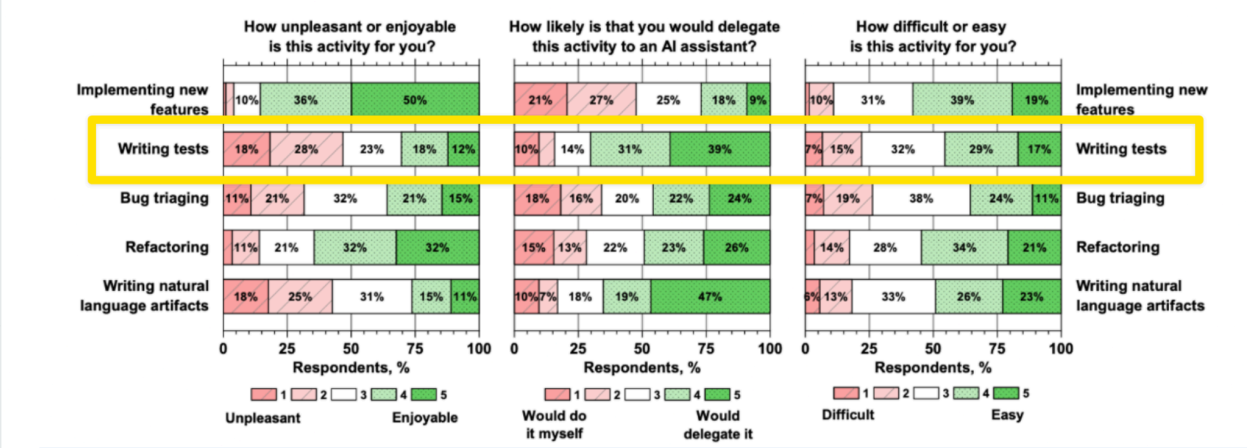
R1: AI Assistant Code Quality Research. GitClear, 2025.

R2: Q3/Q4 2025 Updates: AI Visible Cohorts, Context Engineering Truth. GitClear, 2025

R3: AI Cohort Stats Calculation Overview. GitClear, 2025.

R4: Calculating Diff Delta. GitClear, 2020

R5: Using AI-Based Coding Assistants in Practice: State of Affairs, Perceptions, and Ways Forward. Jetbrains, 2024.



R6: Transcript for Sundar Pichai: CEO of Google and Alphabet | Lex Fridman Podcast #471. LexFridman, 2025.

R7: Typical Git Commit Counts. GitClear, 2024.

R8: Managing Changes in Requirements: An Empirical Investigation. Janes et al, 2013.