State of AI vs. Human Code Generation Report

We found AI-written code produces
~ 1.7x more issues

CodeRabbit

Intro.

Everyone *knew* AI was producing more bugs. *Now* we know what kind.

Over the past year, as AI coding assistants became a standard part of the development workflow, developers have been raising alarms.

Internal dashboards show more late-stage defects, SRE teams report more operational incidents tied to logic and configuration errors, and several high-profile postmortems in 2025 have pointed to AI-authored or AI-assisted changes as contributing factors.

These aren't isolated anecdotes. They reflect a growing pattern. A recent report found that while pull requests per author increased by 20% year-over-year in 2025, thanks to help from AI, incidents per pull request increased by 23.5%.

AI might be accelerating output, but it's also amplifying certain types of mistakes.

A lack of data on what kinds of issues AI creates

Despite this, most organizations still lack hard data on what kinds of issues AI introduces most often or how those patterns differ from human-authored code. That's why we conducted this study: to finally get insight into what kinds of issues teams should be looking for in AI-authored code.

We uncovered the errors AI makes

We analyzed CodeRabbit's reviews of **470 real GitHub open-source pull requests**: 320 of which were labelled as AI co-authored and 150 of which we determined were likely written by human developers (see info on the limitations of our study on the next page).

CodeRabbit's automated review taxonomy looked at logic, maintainability, security, and performance. Every finding was normalized to compare issue rates per 100 PRs.

The results confirm what many engineering teams have been sensing: *AI-generated PRs don't just contain more issues, they contain more of different kinds of issues than human authored PRs.*

And the increased issues skews toward categories that matter most: correctness, readability, error handling, and security.

~1.7x
more issues.

~1.3-1.7x
more critical & major findings.

75%
higher prevalence of logic & correctness issues.

Learn how AI code reviews can help.

10 most notable findings.

01 AI co-authored PRs generate ~1.7× more issues overall.

The average AI PR produced 10.83 findings vs. 6.45 for human PRs.

02 Severity escalates with AI: More critical & major issues.

AI PRs show ~1.4–1.7× more critical and major findings.

03 Logic & correctness problems lead the gap.

Logic/correctness issues were 75% more common in AI PRs.

04 Readability concerns spike more than 3× in AI PRs.

AI code "looks right" at a glance but often violates local idioms or structure.

05 Error handling & exception-path gaps are nearly 2× higher.

AI-generated code often created issues tightly tied to real-world outages.

06 Security issues are ~1.5x higher.

1.56× higher on security issues overall, ~2x with improper password handling.

07 Performance regressions are rare but disproportionately AI-driven.

Excessive I/O operations were ~8× more common in AI PRs.

08 Concurrency & dependency correctness saw ~2× increases.

Incorrect ordering, faulty dependency flow, or misuse of concurrency primitives appeared far more frequently in AI PRs.

09 Formatting problems were 2.66× more common in AI PRs.

Spacing, indentation, structural inconsistencies, and style drift were all more prevalent.

10 AI introduced nearly 2× more naming inconsistencies.

Unclear naming, mismatched terminology, and generic identifiers appeared frequently.

Limitations of our study

Getting data on issues that are more prevalent in AI-authored PRs is critical for engineering teams but the challenge was determining which PRs were AI-authored vs. human authored. Since it was impossible to directly confirm authorship of each PR of a large enough OSS dataset, we checked for mentions that a PR was co-authored by AI and assumed that those that didn't have them were human authored, for the purposes of the study. This resulted in statistically significant differences in issue patterns between the two datasets, so we are sharing that data in this report so teams can better know what to look for. However, we cannot guarantee all the PRs we labelled as human authored were only authored by humans. Our full methodology is shared at the end of the report.

AI co-authored PRs have *more* issues.

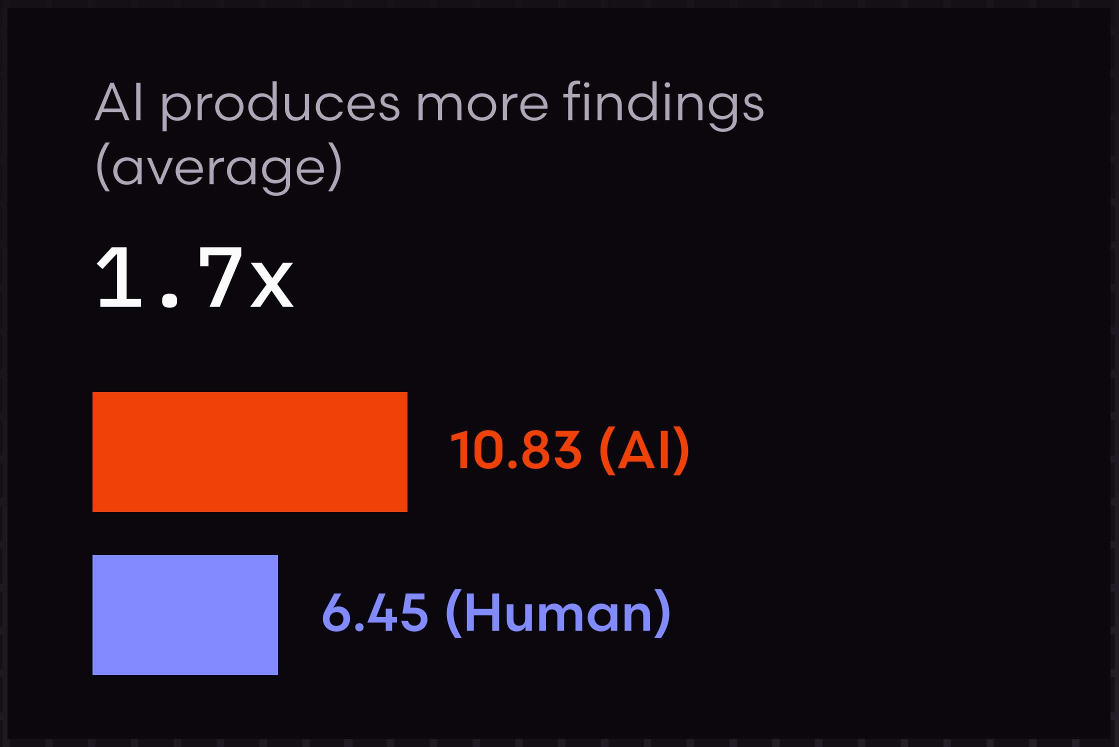
When we compared issue volume across the 470 pull requests, one pattern stood out clearly: AI co-authored PRs contain far more findings than human-only PRs.

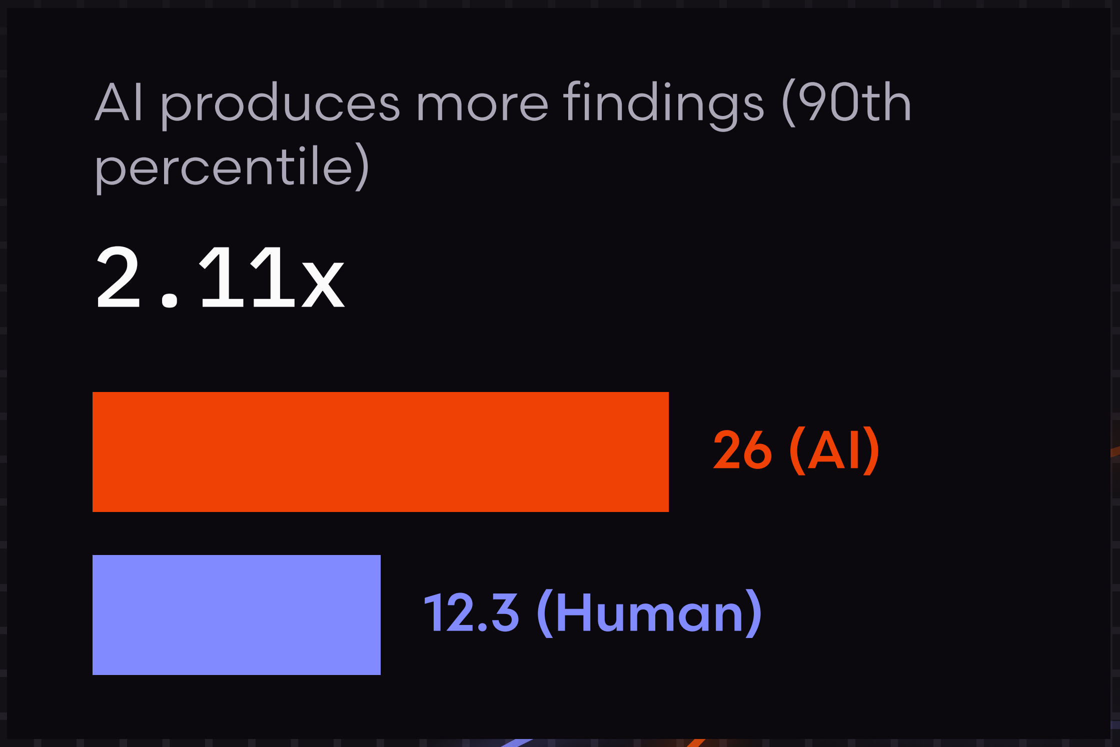
On average, AI PRs have **10.83 issues** per PR or about **1.7× higher** than the **6.45 issues** in human submissions.

AI co-authored PRs also have higher *spikes* in issues.

But the more important story is the distribution. AI-generated PRs have a much heavier tail, meaning they produce far more "busy" reviews.

At the 90th percentile, AI PRs hit 26 issues (vs. 12.3 for humans). At the 95th percentile, the gap widens further with 39.2 for AI vs. 22.65 for humans.





Takeaway: AI PRs are harder to review in multiple ways.

AI doesn't just generate more issues overall, it generates more PRs with a large number of issues, the kind that slow review pipelines and increase defect risk.

Teams adopting AI coding tools should expect higher variance and more frequent spikes in PR issues that demand deeper scrutiny.

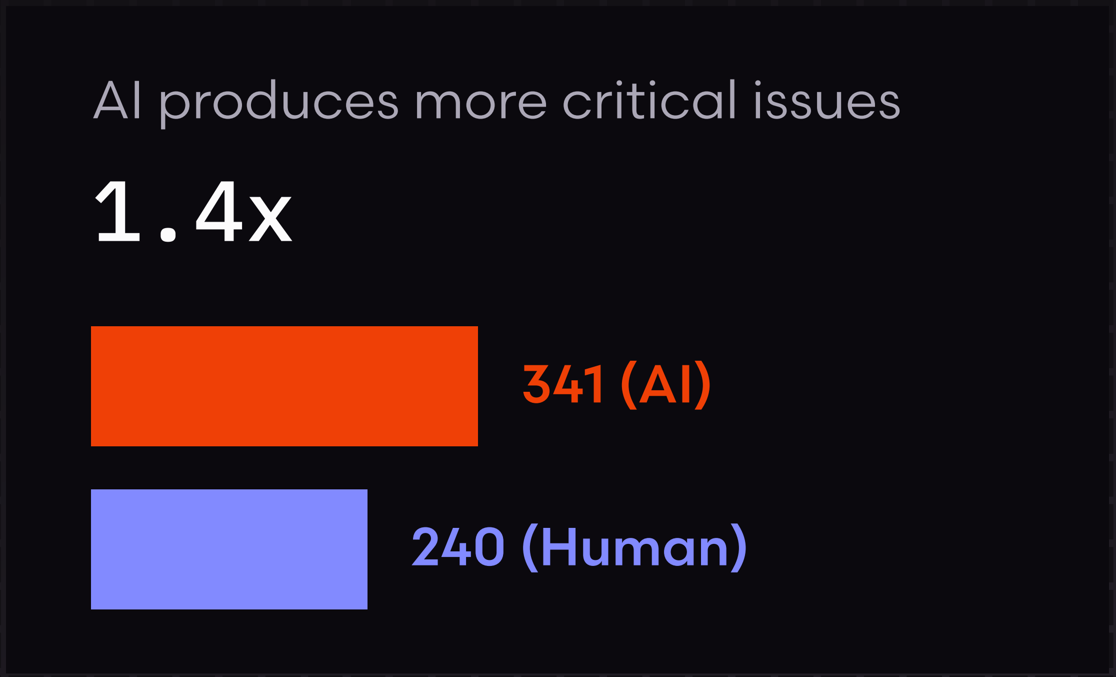
AI PRs *also* have more severe & critical findings.

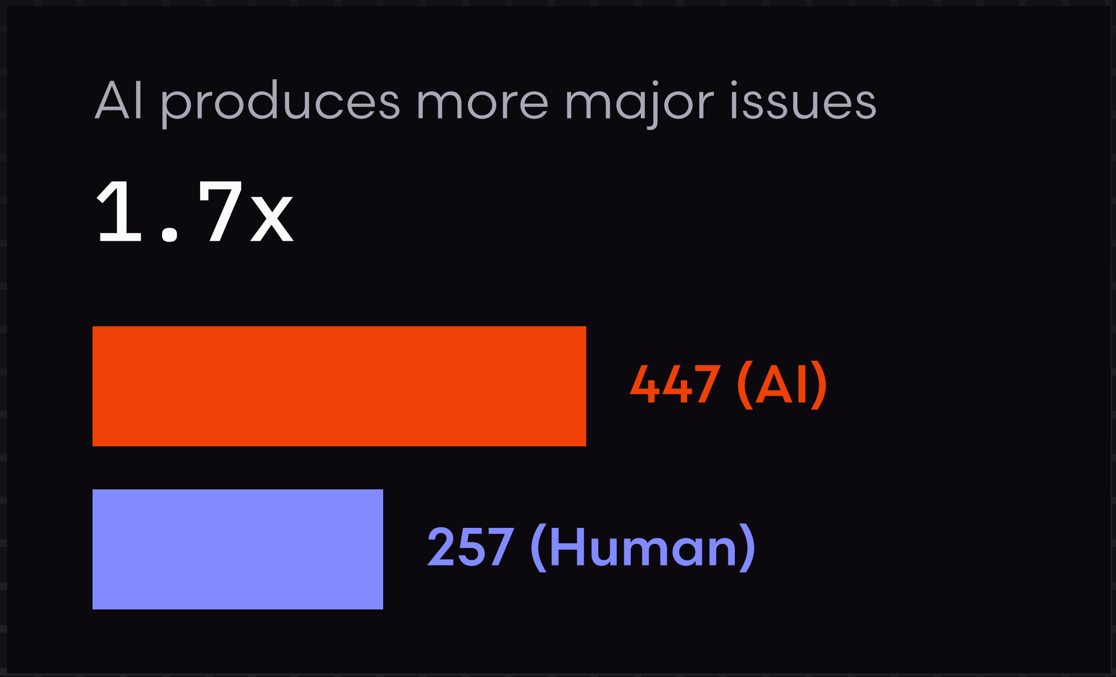
AI co-authored PRs don't just generate more findings, they generate more severe findings.

When normalized per 100 PRs, every severity band is elevated in AI submissions.

- **Critical issues** rise from 240 to 341 (1.4× higher).
- **Major issues** jump from 257 to 447 (1.7× higher).
- **Minor and trivial issues** nearly double.

This pattern reinforces a core theme: AI-authored code increases both the volume and the impact of the issues reviewers must address.





Takeaway: AI-generated PRs have more severe issues and, therefore, more risk.

AI-generated PRs produce more serious defects and pose more risk to production. The increase isn't just noise; it's a meaningful rise in substantive concerns that demand reviewer attention. And could lead to an incident if missed.

AI PR findings are highest around *logic & correctness* issues.

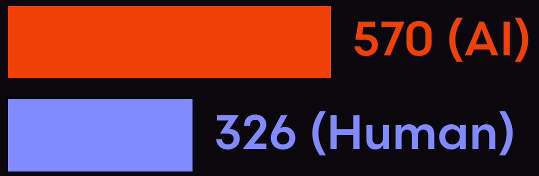
Within every major category including correctness, maintainability, security, and performance, AI co-authored PRs consistently generate more issues than human-only PRs. However, they generate more issues in certain areas.

Logic & correctness.

Business logic errors, misconfigurations, error and exception handling, null-pointer errors, and more.

AI produces more errors

1.75x

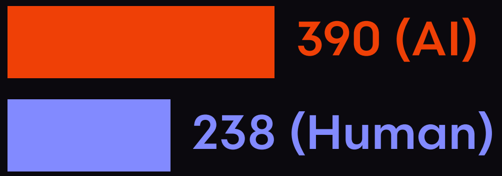


Code quality & maintainability.

Readability, formatting, and naming issues that slow reviews and contribute to long-term technical debt.

AI produces more errors

1.64x

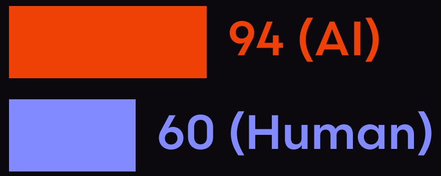


Security findings.

Issues that could lead to incidents like improper credential handling and insecure references.

AI produces more errors

1.57x

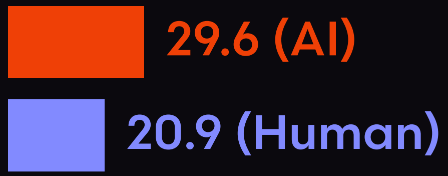


Performance issues.

Performance issues also trend higher (though from a smaller sample size).

AI produces more errors

1.42x



Takeaway: Teams should look for specific types of errors in AI PRs.

AI amplifies every major category of issues, not just overall volume. However, AI is much more likely to create logic and correctness issues than security findings. This information is helpful since teams can better know what to look for when reviewing AI PRs.

Where AI makes *more* mistakes...

So, exactly what kinds of errors is AI making most often? We break them down even further so you know exactly what to look for.

Logic & correctness issues.

Logic and correctness categories show the largest spikes.

- Algorithm and business-logic mistakes appear 194 times per 100 AI PRs versus 86 for humans—over 2.25× higher.
- Error and exception-handling gaps nearly double as well (70 vs. 36, or 1.97x more).
- Misconfigurations, incorrect ordering of operations, and missing dependencies all show similar lifts (around 1.8–1.9× more).
- Concurrency and safety issues stand out too. Incorrect concurrency control is 2.29× more common in AI PRs, and null-pointer/None dereference risks are over 2.2× higher.
- Notably, conditional-logic errors are roughly equal across both groups, one of the only areas where humans and AI stumble at similar rates.

AI error increase and frequency

Algorithm/business logic error	2.25x	194.28 errors
Conditional logic error	1.11x	110.10 errors
Misconfiguration	1.82x	81.14 errors
Error & exception handling issues	1.97x	70.37 errors
Incorrect sequence/dependency	1.81x	26.94 errors
Incorrect concurrency control	2.29x	15.49 errors
Null-pointer/None dereference	2.27x	13.80 errors

Takeaway: Reviewers should focus more on logic & correctness issues.

While AI amplifies all kinds of errors, it particularly amplifies many of the most failure-prone subcategories in modern codebases. That presents additional risk of down time and bugs in production.

Where AI makes *more* mistakes...

Code quality & maintainability challenges.

AI-generated code shows the biggest gaps, not in correctness or security, but in basic code quality, the things that make code readable, maintainable, and easy to review. These issues don't break production, but they slow teams down and compound into long-term technical debt.

- The most striking gap is in readability: AI PRs surface 98 readability issues per 100 PRs, compared to just 31 in human submissions—a 3.15× increase. Formatting problems follow closely, appearing 2.66× more often (59 vs. 22).
- Naming consistency takes a hit as well: AI PRs show 61 unclear naming issues per 100 PRs, nearly double the human rate (humans create 32 or 1.87x less).
- Patterns of unused or redundant code appear 1.64× more frequently in AI-authored changes.

AI error increase and frequency

Code readability	3.15x	97.98 errors
Unclear naming	1.87x	60.61 errors
Code formatting errors	2.66x	59.26 errors
Unused/redundant code	1.64x	33.33 errors

Takeaway: Reviewers shouldn't ignore style & quality issues if they want a maintainable codebase.

AI tends to generate code that looks correct at a glance but violates local conventions. This drives up review time, increases cognitive load for maintainers, and accelerates the accumulation of style-driven technical debt. Not tackling it during review just delays the problem.

Where AI makes *more* mistakes...

Security vulnerabilities.

Security issues show a consistent (and meaningful) lift in AI co-authored PRs. While none of these vulnerabilities are unique to AI-generated code, they appear significantly more often, increasing the risk profile of AI-assisted development.

- The most notable spike is in improper password handling, where AI PRs surface 66 issues per 100 PRs versus 35 for humans, a 1.88× increase. These include hardcoded credentials, unsafe hashing, and ad-hoc authentication logic, all of which create direct exposure paths in production systems.
- Insecure object references appear nearly 2× more often (7.74 vs. 4.05) and injection-style vulnerabilities like XSS show a roughly 2.7× lift.
- Even insecure deserialization, a classic but high-impact flaw, appears ~1.8× more frequently in AI PRs.

AI error increase and frequency

Improper password handling	1.88x	65.99 errors
Insecure object reference	1.91x	7.74 errors
XSS	2.74x	3.70 errors
Insecure deserialization	1.82x	3.70 errors

Takeaway: AI makes dangerous security mistakes that teams need to get better at catching.

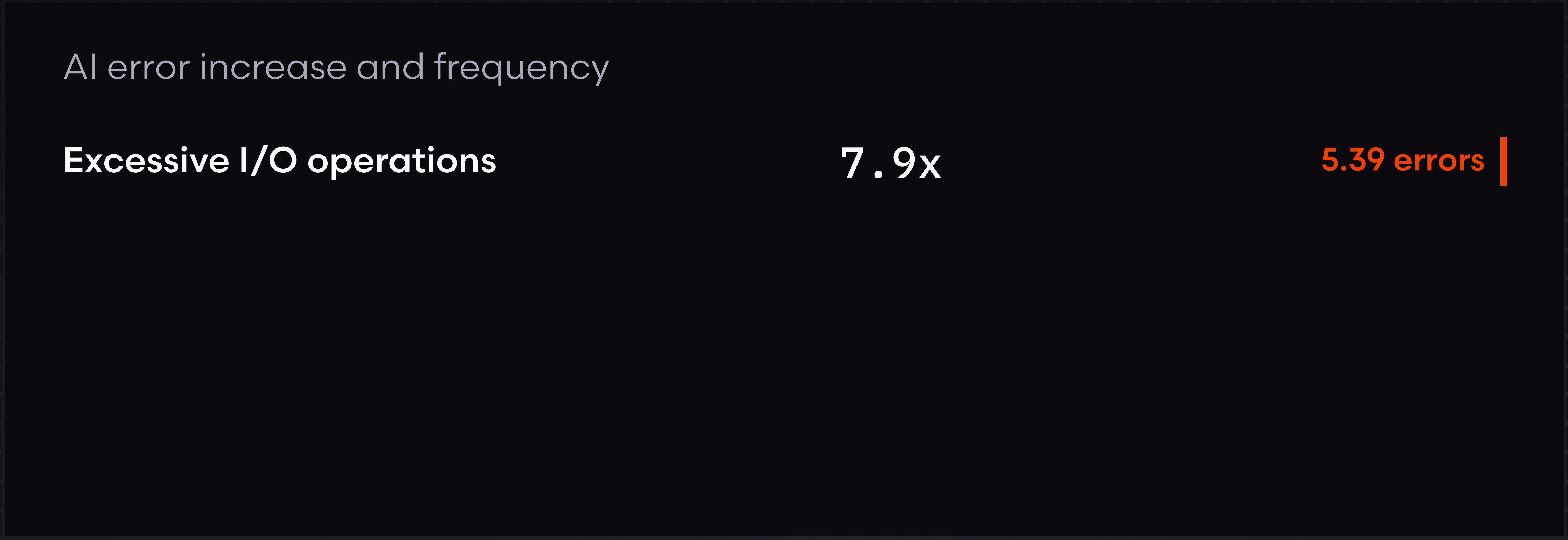
AI amplifies the frequency of foundational security mistakes such as credential handling, unsafe references, injection risks, and more. Teams should consider pairing AI adoption with stronger SAST, linting, and security review practices to ensure their codebase is still protected.

Where AI makes *more* mistakes...

Performance issues.

Performance-related issues are relatively rare overall, but when they do appear, AI-generated code shows a clear pattern: it's less resource-efficient.

- Excessive I/O operations, one of the clearest performance red flags, occur 5.39 times per 100 AI PRs compared to just 0.68 in human PRs. That's a ~7.9× increase. These issues typically manifest as unnecessary file reads, repeated network calls, or unbatched operations that can slow down systems under load.



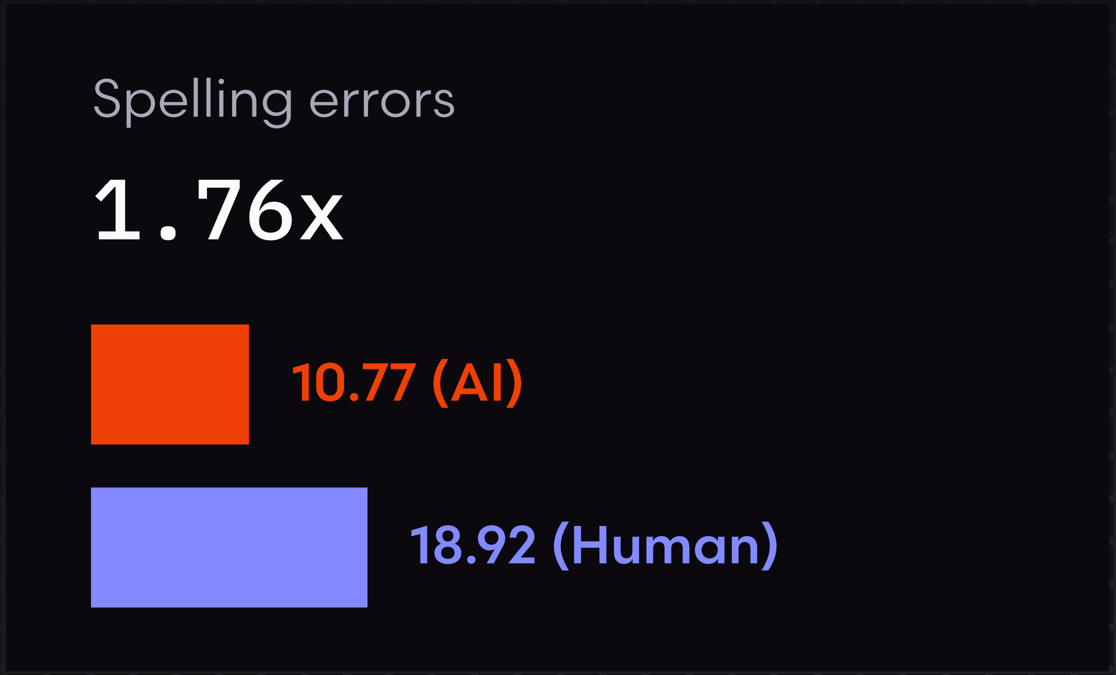
Takeaway: AI-authored code can make your codebase less efficient.

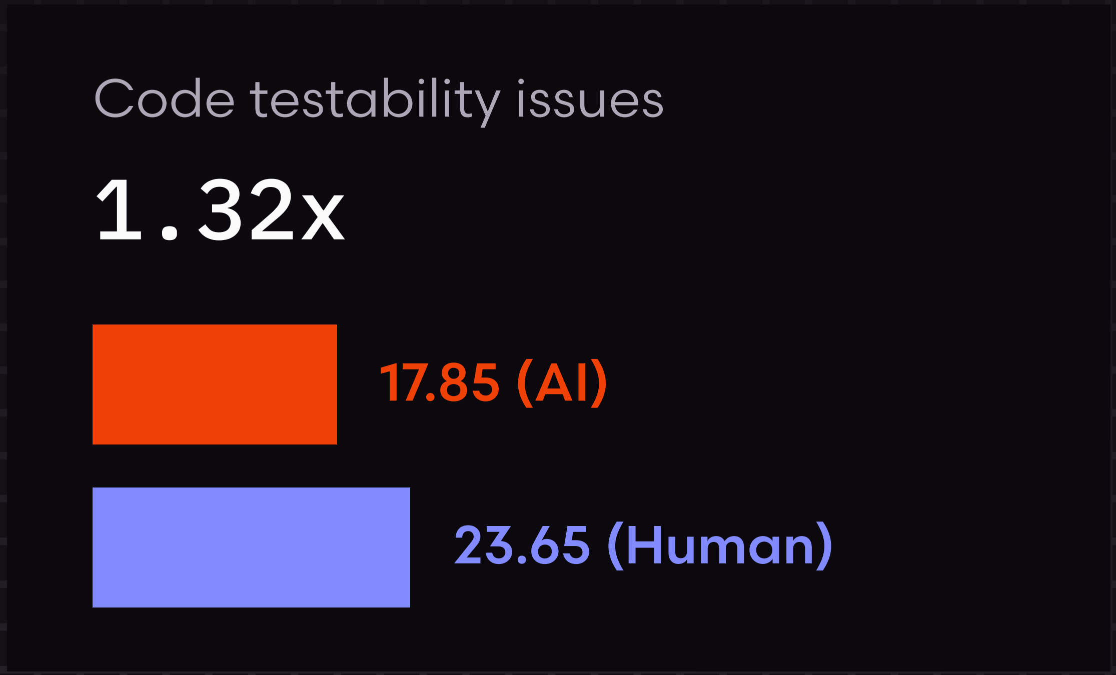
AI is more likely to introduce inefficiency into code. That requires an extra lift from PR authors and reviewers to remediate the issues prior to merging it.

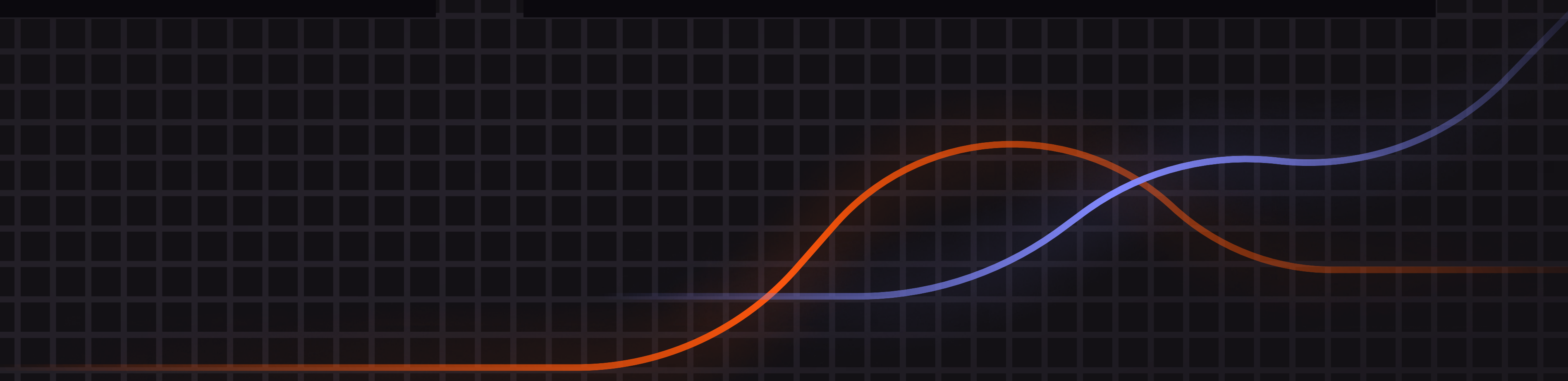
Where AI performs *better* than humans.

Interestingly, a few subcategories lean in the other direction. In these areas, humans make more mistakes than AI.

- **Spelling errors** are almost twice as common in human-authored PRs (18.92 vs. 10.77), perhaps reflecting the fact that humans write far more inline prose and comments. Or that devs are just bad at spelling? 😂
- **Testability issues** also appear more often in human code (23.65 vs. 17.85).







Takeaway: Developers are only... human.

Developers aren't all experts at spelling, which won't surprise anyone who's reviewed a PR in their lifetime. But humans also write more descriptive comments, documentation, and tests. That tends to lead to more spelling issues and slightly higher testability findings on human PRs.

Why these AI patterns emerge.

01 Local business logic & domain context.

LLMs can generalize patterns from broad training data but often miss project-specific invariants, configuration rules, and edge-cases. This drives higher rates of algorithmic errors and misconfigurations in AI co-authored PRs.

02 Surface-level correctness without deep control-flow safety.

AI-generated code can look structurally correct while omitting essential safeguards such as null checks, short-circuit conditions, and complete exception paths. This aligns with the elevated error-handling gaps and null-pointer issues.

03 Inconsistent adherence to repository conventions.

Models revert to generic naming, formatting, and structural patterns rather than local style conventions. This shows up in the significant increase in readability, naming, and formatting findings.

04 Security best-practice drift.

Without explicit constraints, LLMs may reproduce outdated or unsafe patterns from older code, such as improper password handling or insecure references. This maps cleanly to the rise in security vulnerabilities across AI PRs.

05 Naïve resource usage.

Models default to clarity over efficiency unless prompted otherwise, leading to more excessive I/O, repeated operations, or suboptimal data structures.

Takeaways & mitigations.

As AI-generated code becomes a standard part of the development workflow, teams need guardrails that counter the specific weaknesses surfaced in the data.

Here we give you a roadmap to help you ensure that your AI's speed doesn't amplify defects downstream.

Provide up-front project context.

- Give models access to project-specific constraints (invariants, config patterns, architectural rules).
- Use prompt snippets or repository "capsules" to ground the model in local domain logic.

Enforce style with policy-as-code.

- Apply strict CI rules for formatting, naming, and structure (formatters, linters, style guides).
- This directly reduces readability, formatting, and naming issues, some of the largest AI defect categories.

Add correctness safety rails.

- Require pre-merge tests for any non-trivial control flow, including negative and edge cases.
- Use nullability/type assertions at module boundaries.
- Standardize error-handling patterns (don't swallow exceptions; centralize handlers).

Codify security defaults.

- Centralize credential/password handling and forbid ad-hoc approaches.
- Use security linters and SAST checks for unsafe references, deserialization, and XSS.

Guide performance behavior.

- Encourage idiomatic data structures, batched I/O, and pagination.
- Add smoke tests for I/O-heavy or resource-sensitive paths.

Adopt AI-aware PR checklists.

Include targeted questions such as:

- "Are error paths tested?"
- "Are concurrency primitives correct?"
- "Are passwords handled via the approved helper?"

Get a third-party AI code review tool.

- AI code reviews can do a first-pass to find most of the issues and do the heavy lifting of remediating AI-authored PRs.
- Use a context-rich, third-party tool like CodeRabbit that maps your codebase and brings in dozens of points of context.
- Don't trust the AI that created the error to find it. If it added the error into your code, it's less likely to find it.

Methodology.

Data sources.

All findings are based on a scrape of open-source GitHub pull requests (PRs) that used CodeRabbit for reviews.

PRs were classified as AI co-authored when tools such as Claude, Cursor, or Codex explicitly indicated authorship.

Sample size.

320 AI co-authored PRs and 150 human-only PRs were included in the analysis.

320 (AI)

150 (Human)

Filtering & preprocessing.

A small number of PRs incorrectly marked as AI co-authored `(has_ai_coauthors = FALSE)` were removed from the AI group.

Classification types were assigned by injecting CodeRabbit review comments tagged with the taxonomy categories.

Normalization.

`Issues per 100 PRs`

How we expressed all frequency metrics to enable apples-to-apples comparisons across cohorts.

Statistical approach.

- Differences between groups were quantified using Poisson rate ratios with 95% confidence intervals.
- CIs were reported for all top-level categories and subcategories with ≥20 total occurrences.

Limitations

- Results reflect CodeRabbit's taxonomy and coverage, which may emphasize certain issue types.
- Some PRs that didn't explicitly indicate AI-assistance might have included AI in their creation. For the purposes of this study, we assumed they did not and found statistically significant findings classifying them in this way.
- Repository mix, code domains, and PR selection criteria may introduce bias into category frequencies.

About CodeRabbit.

CodeRabbit is the category-defining platform for AI code reviews, built for modern engineering teams navigating the rise of AI-generated development.

By delivering context-aware reviews that pull in dozens of points of context, CodeRabbit provides the most comprehensive reviews coupled with customization features to tailor your review to your codebase and reduce the noise.

Trusted by thousands of companies and open-source projects worldwide, CodeRabbit helps organizations catch bugs, strengthen security, and ship reliable code at speed.

Learn more at www.coderabbit.ai.

CodeRabbit