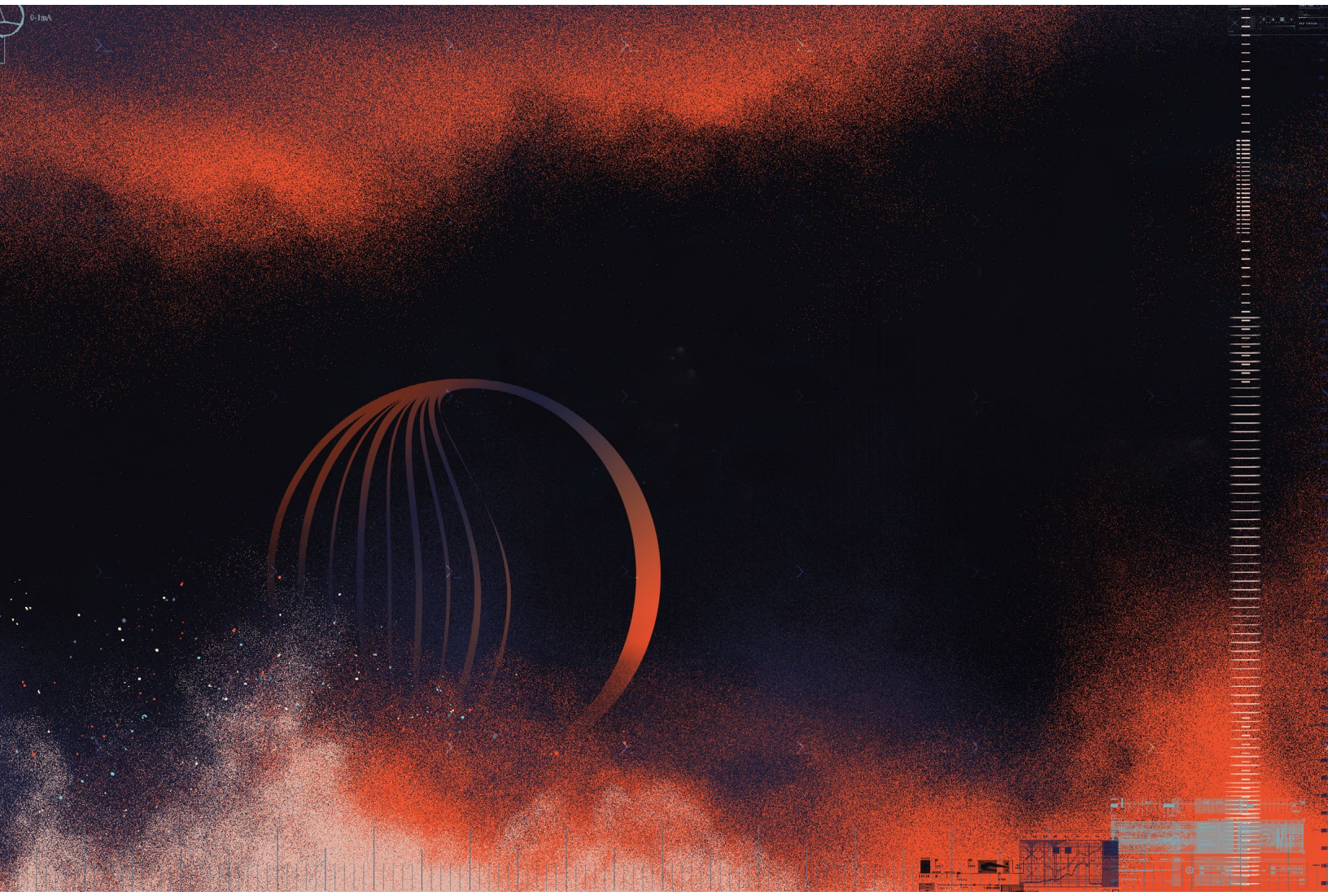
signal65



Evaluating AI Code Review Tools: A Real-World Bug Detection Study

AUTHOR

Mitch Lewis
Performance Analyst | Signal65

IN PARTNERSHIP WITH

CodeRabbit

MARCH 2026

Executive Summary

AI code review tools are becoming a critical component of modern software development. Generative AI has accelerated code production dramatically, but this speed introduces new challenges: ensuring safety, reliability, and maintainability at scale. AI-driven code review presents a clear opportunity to meet this challenge — but only when tools can accurately identify real issues without flooding developers with false positives or low-value noise.

To evaluate the current landscape, Signal65 conducted a hands-on assessment of five AI code review tools, each tested against bug-introducing pull requests across six open source repositories. CodeRabbit emerged as the leading solution, with three standout advantages:

Superior Critical Bug Detection
CodeRabbit identified the most high-severity bugs of any tool evaluated

High Precision
95.88% precision, minimizing false positives while surfacing real, impactful issues

Consistent Cross-environment Performance
Led in critical bug detection in 5 of 6 repositories and produced the fewest incorrect findings in 4 of 6

Test Overview

Signal65 tested five AI code review tools to assess their ability to detect real bugs and avoid false positives and unnecessary noise. The tools evaluated were:

- CodeRabbit
- Cursor BugBot
- GitHub Copilot
- Greptile
- Qodo Merge

Methodology

Testing used real historical bugs from six open source repositories spanning distinct programming languages. For each repository, ten bug-introducing pull requests were identified from the project's history, then recreated by rewinding a branch to just before the bug was introduced. Each tool reviewed the same pull requests in isolated repositories, all running default settings.

Repository	Language	Description
vLLM	Python	LLM inference and serving engine
ElasticSearch	Java	Distributed RESTful search engine
Axios	JavaScript	Promise-based HTTP client
Next.js	TypeScript	React framework
Cilium	Go	eBPF-based networking and security
Puma	Ruby	Parallel Ruby/Rack web server

Figure 1: *Open Source Repositories*

All reviews were manually graded by Signal65 analysts against a pre-defined rubric. A finding only counted as a bug if it included an inline comment referencing specific lines of code. Bugs identified by multiple tools received consistent ratings across all tools.

Rating	Criteria
Critical	Crash, security vulnerability, data loss, incorrect results in common usage, broken API contract, hotfix-worthy
Moderate	Edge case failures, performance degradation, non-fatal user-visible malfunction
Minor	Low-impact edge cases, incorrect logging, misleading messages, limited maintainability issues
Incorrect	False positives, or trivial suggestions that don't meaningfully improve the code

Figure 2: *Bug Severity Rubric*

Findings

Results are evaluated across three metrics:

- True positives: accurately identified bugs
- False positives: incorrect or misleading findings
- Precision: the ratio of true to false positives

Together, these provide a complete picture of each tool's real-world utility.

Overview

CodeRabbit delivered the strongest overall performance, achieving consistently high results across all three metrics. It identified the second highest number of true positives (93), recorded the second fewest false positives (4), and achieved 95.88% precision — near the top in every dimension evaluated.

Cursor BugBot achieved the highest precision (95.95%) and the fewest false positives (3), but found 23% fewer true positives than CodeRabbit and 28% fewer critical bugs, limiting its overall value. Qodo Merge led in raw true positive count (129) but generated 7.5x more false positives than CodeRabbit and achieved only 81.13% precision. GitHub Copilot produced the most false positives of any tool (41) at 64.35% precision. Greptile achieved the third highest precision (86.36%), but found the fewest total true positives of any tool evaluated.

When combining the ability to find real bugs, avoid false alarms, and surface critical issues, CodeRabbit stands apart as the most consistent, reliable, and highest-impact tool in this evaluation.

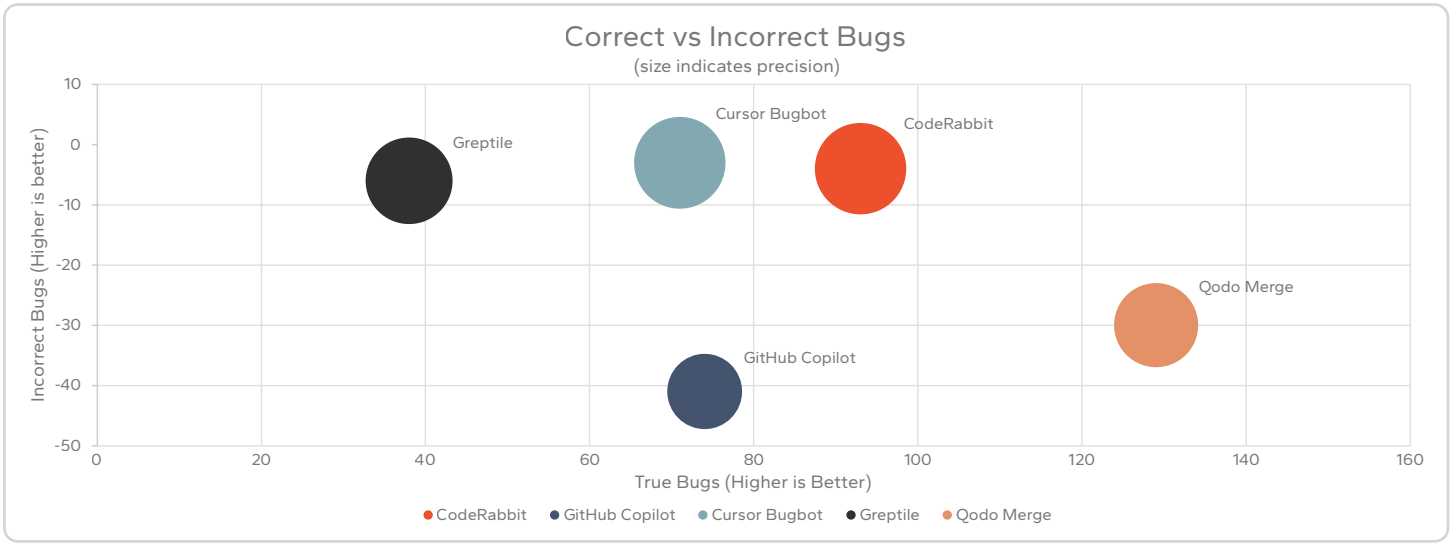


Figure3: *Correct vs Incorrect Bugs*

Note: False positives are represented as negative values on the y-axis.

True Positives: Accurately Identified Bugs

True positives reflect a tool's core capability — its ability to find real bugs that developers should act on. Qodo Merge led all tools with 129 true positives, followed by CodeRabbit with 93. GitHub Copilot identified 74, Cursor BugBot identified 71, and Greptile identified the fewest with 38.

However, raw true positive counts alone do not tell the full story. Tools that flag more issues overall will naturally surface more true positives — but at the cost of also generating more noise. Identification of false positives and overall precision, examined below, accounts for this tradeoff.

False Positives: Incorrect or Misleading Findings

False positives are the most disruptive output an AI code review tool can produce. They waste developer time, erode trust, and can lead to unnecessary or harmful code changes. Cursor Bugbot recorded the fewest false positives with just 3, followed closely by CodeRabbit with 4. Greptile produced 6 false positives. Qodo Merge and GitHub Copilot generated the most noise, with 30 and 41 false positives respectively.

The gap between the top and bottom performers is significant. GitHub Copilot produced more than 10x the false positives of CodeRabbit, and Qodo Merge produced 7.5x more.

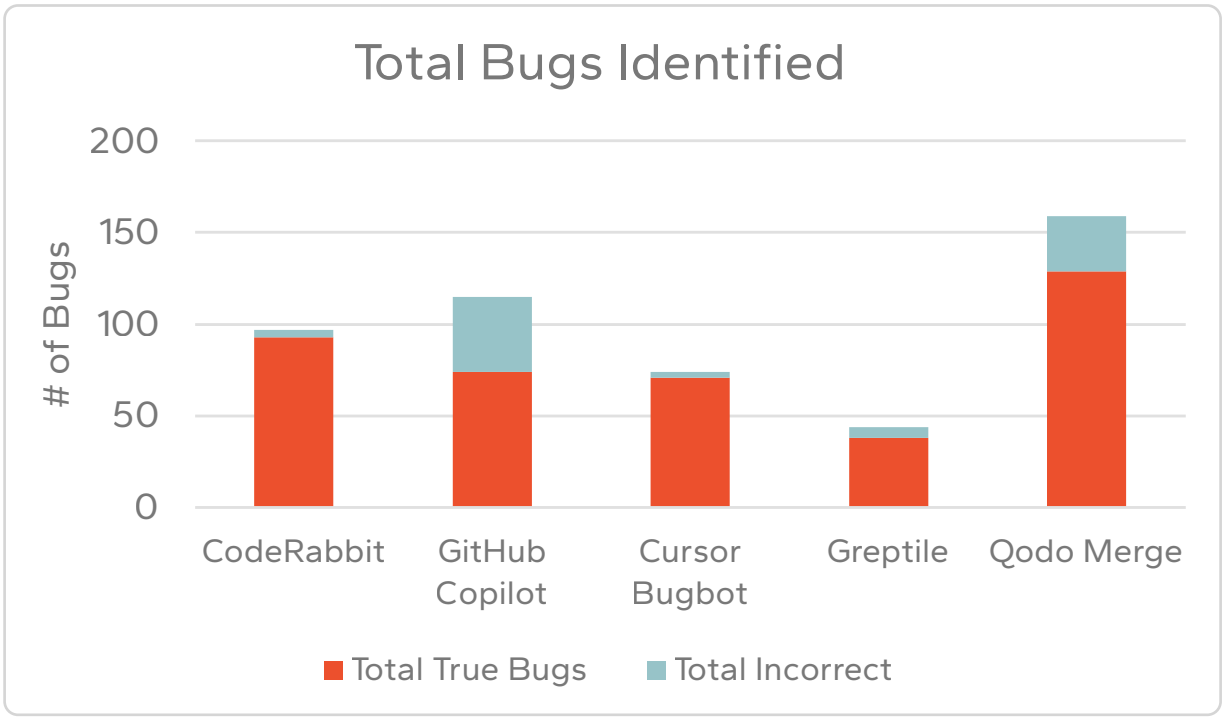


Figure 4: *Total bugs Identified*

True-to-False Positive Ratio: Precision

Precision — the share of flagged issues that are genuinely valid — determines how much a developer can trust a tool's output. High precision means developers can act on findings with confidence; low precision means sorting through noise to find what matters.

CodeRabbit and Cursor BugBot were nearly tied at the top, achieving 95.88% and 95.95% precision respectively. All three remaining tools fell considerably short, ranging from 64.35% to 86.36%.

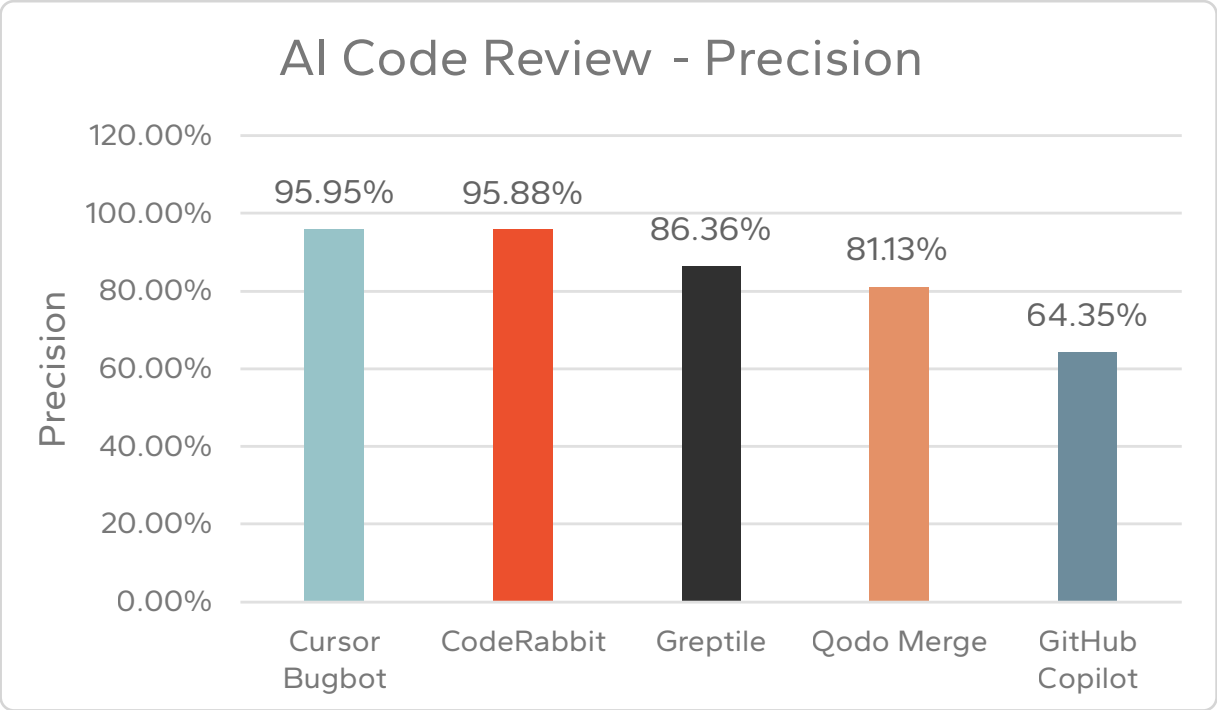


Figure 5: *AI Code Review Precision*

Critical Bug Detection

Beyond overall bug counts, the severity distribution of findings reveals how much practical value each tool delivers. Critical bugs — those that can cause crashes, security vulnerabilities, data loss, or incorrect results — represent the highest-value output of any code review tool.

In addition to achieving leading precision, CodeRabbit identified the most critical bugs with 25. No other tool achieved competitive results across both precision and critical bug detection simultaneously.

Qodo Merge found the second most critical bugs (22), but its low precision (81.13%) and high false positive count significantly undercut that result. Cursor BugBot achieved nearly identical precision to CodeRabbit, but found 28% fewer critical bugs.

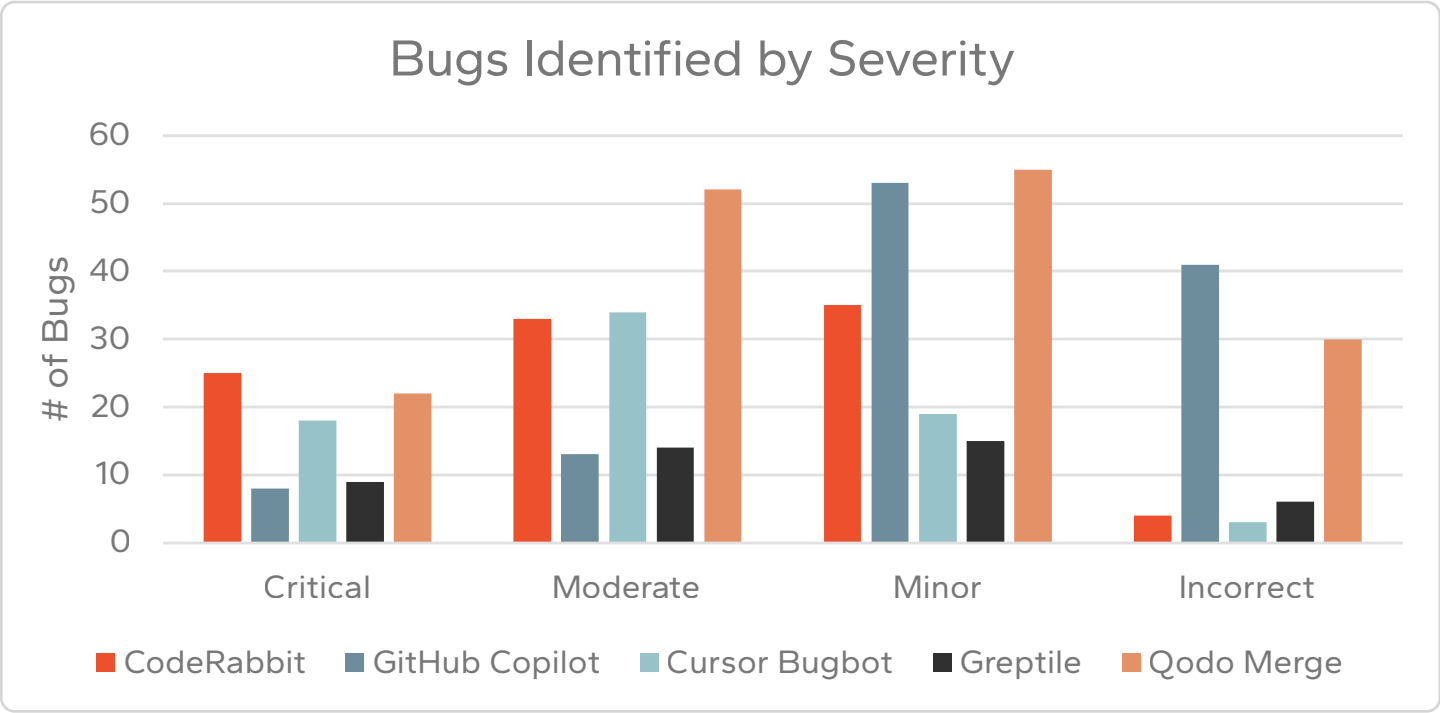


Figure 6: *Bugs by Severity*

Lower-severity issues still provide value, but their overall impact is substantially lower. Minor bugs are often limited to formatting or readability concerns and typically do not affect functionality. In practice, over-reporting such low-impact issues can introduce noise and reduce developer trust in the tool. Both Qodo Merge and GitHub Copilot increased their total finding counts largely through a high volume of minor bugs, a pattern that also correlates with increased false positives.

Signal65 Comment *– One noteworthy distinction is that CodeRabbit reports its most minor comments under a specific "nitpick comments" dropdown menu, which is separated from the inline comments. This functionality provides a clean approach that enables very minor issues to be reported, without adding additional noise to the review. Since the nitpick comments are hidden by default, and not included as inline comments, they were not graded as bugs for this evaluation and are not reflected in CodeRabbit's total bug count.*

Consistency Across Repositories

A tool's value compounds when it performs reliably across different languages and codebases — not just in isolated cases. CodeRabbit demonstrated exactly this, leading in critical bug detection in five of six repositories and generating the fewest incorrect findings in four of six.

This consistency is significant. It indicates that CodeRabbit's performance is not tied to a specific language or codebase type, but reflects a broad capability applicable to real-world development environments. Detailed per-repository results can be found in the appendix.

Repository	Most Critical Bugs Identified	Fewest Incorrect Issues
vLLM	✔	✔
ElasticSearch	✔	
Axios		✔
Next.js	✔	✔
Cilium	✔	
Puma	✔	✔

Figure 7: *CodeRabbit Performance by Repository*

High Signal, Low Noise

Beyond raw bug detection, AI code review tools also differ in how they present information — summaries, diagrams, security overviews, inline comments, and more. The right balance adds context without overwhelming developers; too much output becomes noise, while too little leaves findings unexplained.

Several tools evaluated took notably different approaches. Some produced extensive multi-paragraph summaries with charts and diagrams, which may suit certain teams but can feel excessive in fast-moving development workflows. Others erred in the opposite direction, providing little context alongside a finding, making the cause and impact difficult to assess.

CodeRabbit demonstrated a balanced approach. Summaries were generally accurate and concise. Inline bug comments included relevant code excerpts alongside clear explanations. Lower-priority items — minor nitpicks, walkthrough notes, and supplementary information — were collapsed behind dropdowns, keeping the primary review surface clean without discarding useful detail. While user experience is highly subjective, Signal65 found this approach to be among the most developer-friendly of the five tools evaluated, and believes it contributes meaningfully to CodeRabbit's overall signal-to-noise advantage.

Additional Features

While not the focus of this evaluation, several CodeRabbit capabilities were noted by Signal65 analysts as particularly valuable complements to its core review performance:

- **Issue Ticket Analysis** – For pull requests linked to Jira or GitHub issues, CodeRabbit analyzes the ticket and assesses whether the pull request actually addresses it.
- **Built-in Chat** – Developers can reply directly to CodeRabbit comments to ask clarifying questions, helping teams quickly assess the severity and validity of a finding.
- **Codegraph Analysis** – CodeRabbit builds a graph representation of the codebase to deepen its understanding of code intent and cross-file dependencies, enabling it to surface bugs that simpler tools miss.
- **MCP Tool Integration** – CodeRabbit supports external MCP servers (including Sentry, Linear, and Context7), allowing it to incorporate logs, issue history, and technical documentation into its reviews.

Signal65 Comment *– These features are notable for their ability to further enhance AI code review capabilities beyond the results captured in this study. Competitive tools may additionally include some subset of these features, but Signal65 believes that combination of this functionality alongside CodeRabbit's quantitative results reinforces its position as the leading solution in this evaluation.*

Final Thoughts

AI code review is becoming a necessity as development teams contend with growing code volume and the quality variability introduced by AI-generated code. This study evaluated five prominent tools by recreating 60 historical bugs across six open source repositories — a methodology designed to reflect real-world conditions rather than synthetic benchmarks.

CodeRabbit emerged as the clear leader. It identified more critical bugs than any other tool, achieved 95.88% precision, and delivered consistent results across every repository and language tested. Where other tools frequently traded accuracy for volume — or sacrificed critical bug detection for lower noise — CodeRabbit demonstrated that high precision and high-impact detection are not mutually exclusive.

Throughout this evaluation, Signal65 found CodeRabbit to offer a strong combination of accuracy, coverage, and consistency. It consistently identified a large number of critical bugs while maintaining high precision across multiple codebases. These characteristics suggest it can be a practical and reliable option for development teams seeking dependable results from AI-assisted code review tools.

Appendix

Figure 8 provides an overview of all bugs identified in this evaluation.

Tool	Critical	Moderate	Minor	Incorrect	Precision
CodeRabbit	25	33	35	4	95.88%
GitHub Copilot	8	13	53	41	64.35%
Cursor Bugbot	18	34	19	3	95.95%
Greptile	9	14	15	6	86.36%
Qodo Merge	22	52	55	30	81.13%

Figure 8: *CodeRabbit Performance by Repository*

vLLM Repository Results

The reviews of vLLM code measured the ability of each code review tool to review a primarily Python-based repository. Just as in the overall results, CodeRabbit distinguished itself as a tool capable of identifying a high level of critical bugs, while keeping noise and incorrect responses to a minimum.

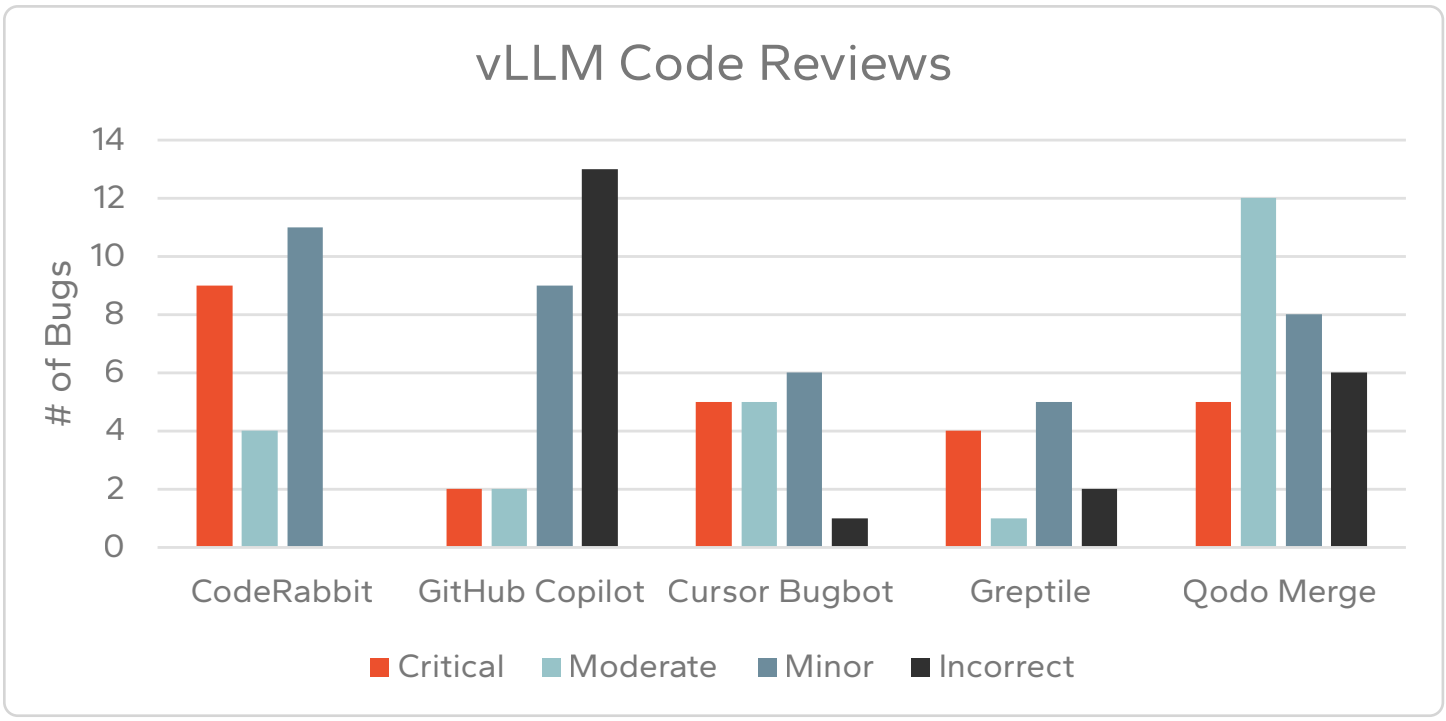


Figure 9: *vLLM Repository Results*

Across all vLLM code reviews, CodeRabbit found 24 total bugs, all of which were determined to be valid issues. Notably, CodeRabbit was the only tool to not identify any incorrect issues for this repository. Qodo Merge was the only tool that correctly identified more bugs, with 25, however it additionally identified 6 incorrect issues. Among the 25 issues identified by CodeRabbit, 9 were critical, significantly leading all tools.

ElasticSearch Repository Results

Code reviews of the ElasticSearch repo tested each tool's ability to review primarily Java-based code. In this repository, CodeRabbit again achieved the highest number of critical bugs identified, tying Qodo Merge with 5. Notably, however, Qodo Merge also identified 5 incorrect issues. While CodeRabbit did falsely identify two issues in this repository, it was tied for the second lowest number of incorrect issues alongside Greptile. CodeRabbit additionally found the second highest number of bugs overall, tying Cursor BugBot with 18.

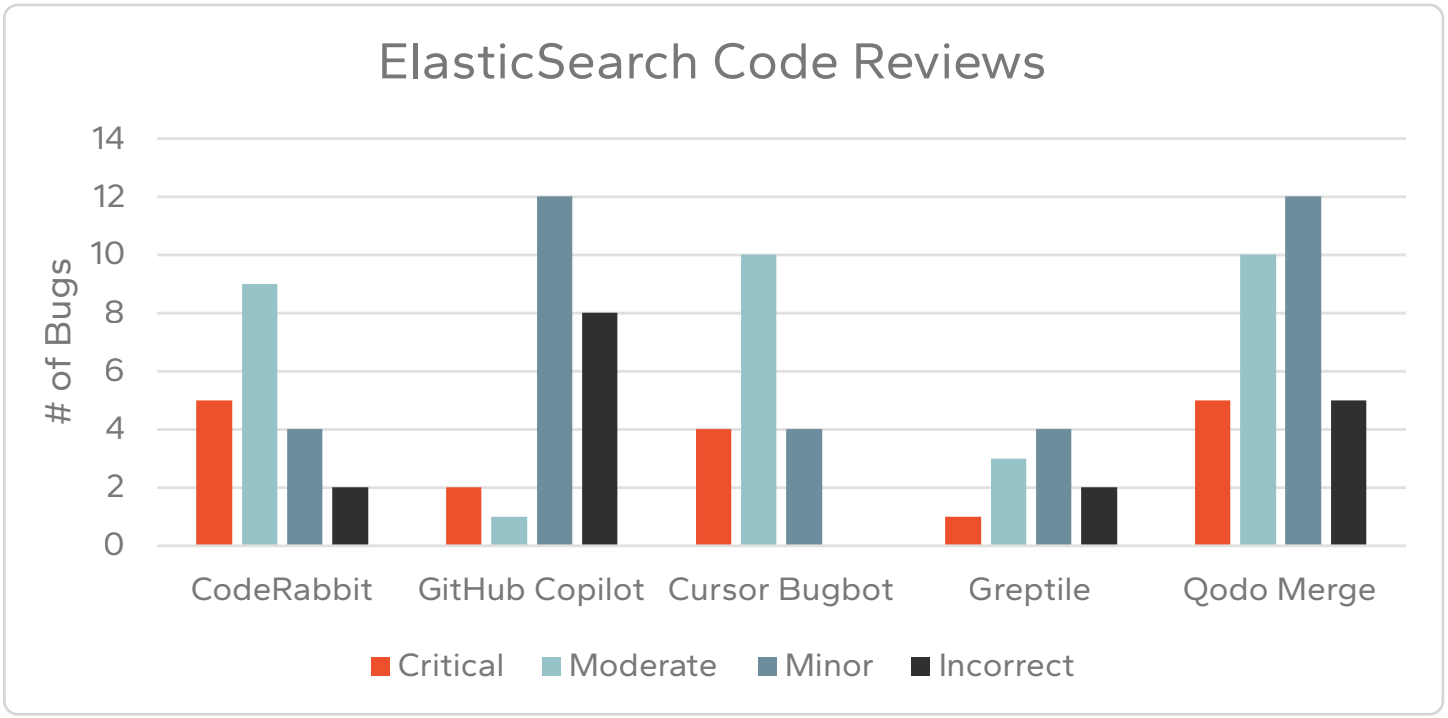


Figure 10: *ElasticSearch Repository Results*

Axios Repository Results

The Axios repository was utilized to evaluate code review in primarily JavaScript-based code. In this repository, CodeRabbit found neither the most overall bugs, nor the most critical, however, it still demonstrated its value through accuracy and restraint. CodeRabbit was the only tool to record zero incorrect responses.

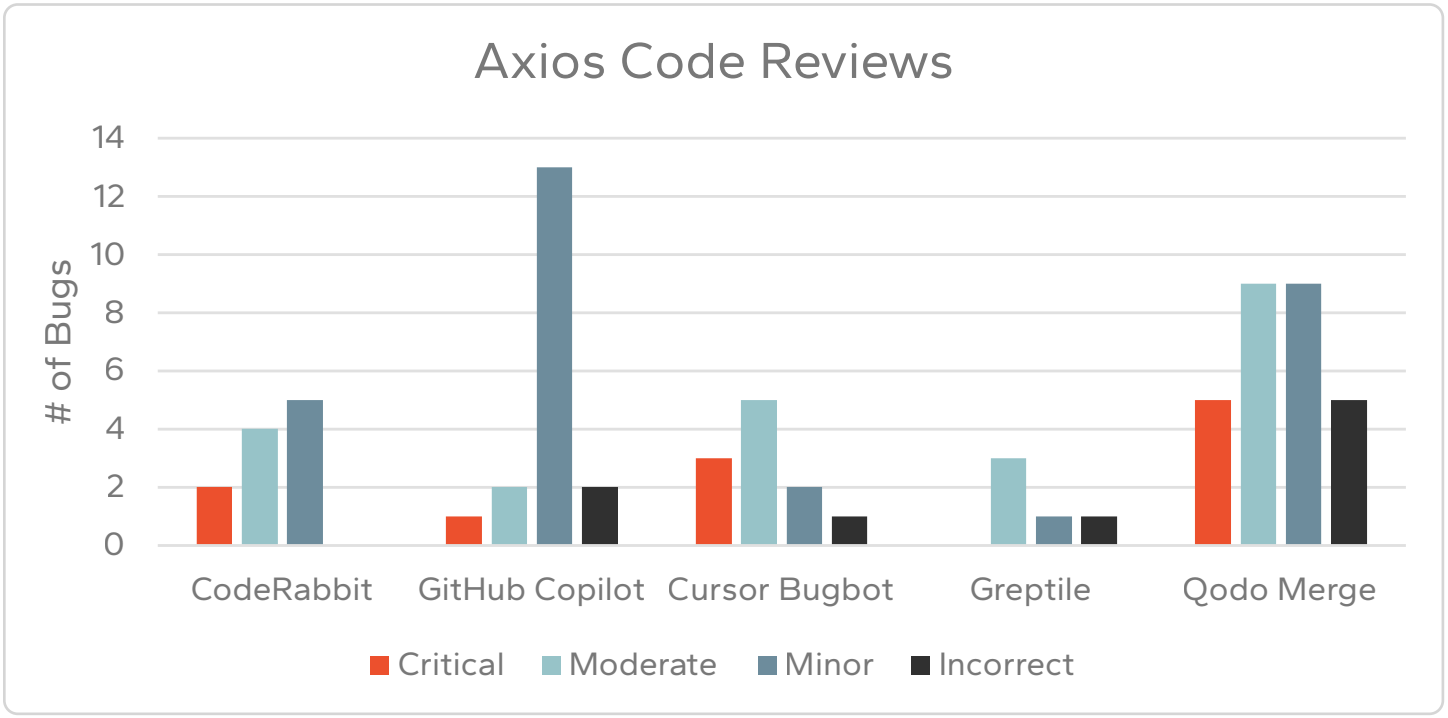


Figure 11: *Axios Repository Results*

Next.js Repository Results

The Next.js repository was utilized to evaluate code reviews of TypeScript code. In this repository, CodeRabbit again stood out amongst the tools tested. CodeRabbit accurately identified the most bugs overall. While a large portion of these bugs were classified as minor, CodeRabbit, also identified the most critical bugs, tied with Qodo Merge. This demonstrates CodeRabbits ability to identify both minor issues alongside far more critical concerns. Notably, CodeRabbit did not identify any incorrect issues, again highlighting its accuracy and restraint.

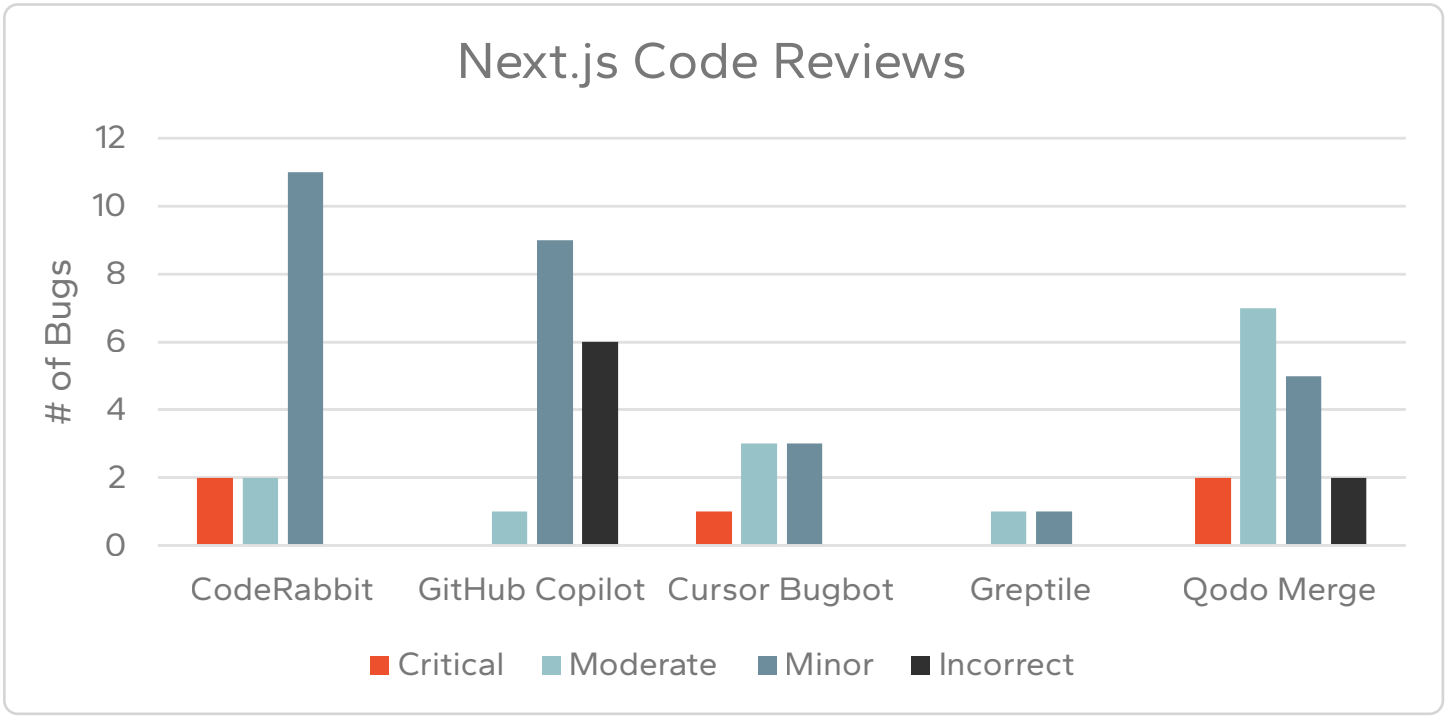


Figure 12: *Next.js Results*

Cilium Repository Results

The Cilium repository was utilized to evaluate code reviews on predominantly Go-based code. In this repository, CodeRabbit was one of three tools to correctly identify 5 critical bugs, and it maintained its low-level of incorrect responses, only generating one.

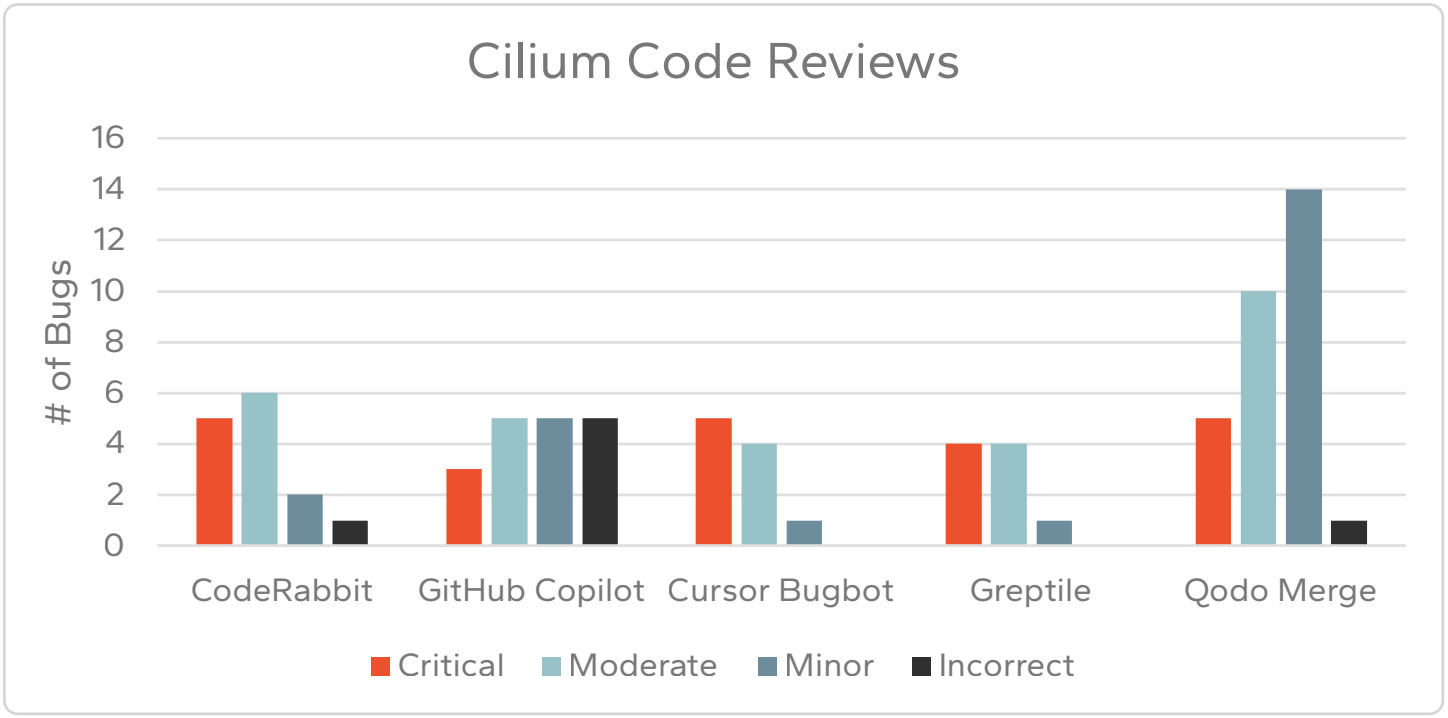


Figure 13: *Cilium Results*

Puma Repository Results

Puma, the final repository evaluated, was utilized to evaluate code reviews of Ruby code. For this repository, CodeRabbit accurately identified the most bugs, as well as the most critical bugs. Notably, CodeRabbit was the only tool evaluated that identified any critical bugs across the ten pull requests in the Puma repository.

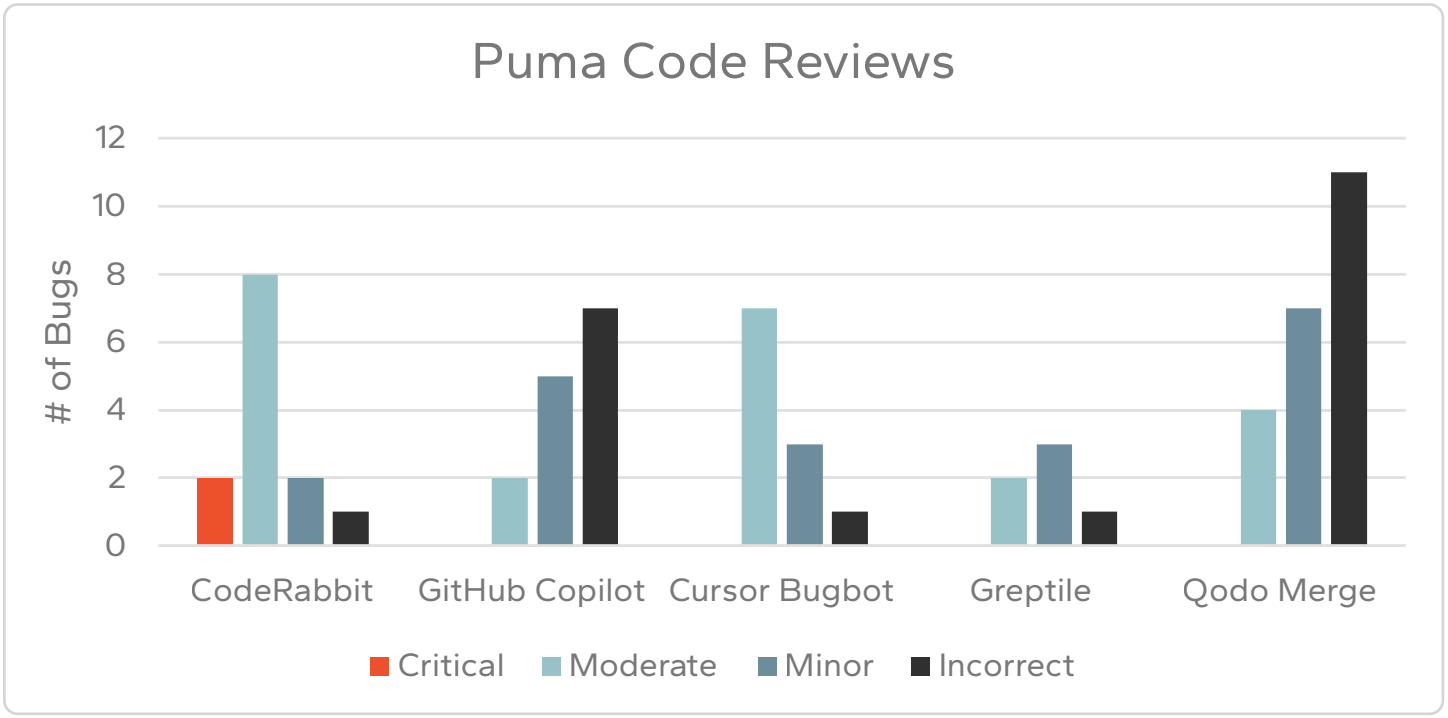


Figure 14: *Puma Results*

Important Information About this Report

CONTRIBUTORS
Mitch Lewis
Performance Analyst | Signal65

PUBLISHER
Ryan Shrout
President and GM | Signal65

INQUIRIES
Contact us if you would like to discuss this report and Signal65 will respond promptly.

CITATIONS
This paper can be cited by accredited press and analysts, but must be cited in-context, displaying author's name, author's title, and "Signal65." Non-press and non-analysts must receive prior written permission by Signal65 for any citations.

LICENSING

DISCLOSURES
Signal65 provides research, analysis, advising, and consulting to many high-tech companies, including those mentioned in this paper. No employees at the firm hold any equity positions with any companies cited in this document.

ABOUT SIGNAL65
Signal65 is an independent research, analysis, and advisory firm, focused on digital innovation and market-disrupting technologies and trends. Every day our analysts, researchers, and advisors help business leaders from around the world anticipate tectonic shifts in their industries and leverage disruptive innovation to either gain or maintain a competitive advantage in their markets.

CONTACT INFORMATION
Signal65 I signal65.com