ChatGPT

# AI-Assisted Coding, Bug Risk, and Safe Efficiency Gains

## Executive summary

Across the strongest evidence published between **2025-11-14 and 2026-05-14**, AI-assisted coding looks less like a simple productivity win and more like a **throughput multiplier with a reliability tax**. Recent repository studies, enterprise telemetry, and industry surveys consistently show that AI can raise output, PR volume, and release cadence, but it also tends to increase downstream defects, review burden, deployment remediation, and long-lived technical debt unless teams strengthen review, testing, and release controls at the same time. The clearest recent signals are Faros AI's telemetry data showing **+28% bugs per PR, 5x median review time, 3x incidents per PR, and 10x code churn**; Harness' large-enterprise survey showing the heaviest AI-coding users report **more deployment problems, more rollbacks/hotfixes/customer-impacting incidents, and longer MTTR**; academic work on Cursor adoption showing **transient velocity gains followed by higher warnings and complexity**; and large-scale empirical studies showing AI-authored commits introduce issues that often persist at the repository head for months. [1]

The efficiency upside is also real. Sonar's 2026 developer survey reports an average **self-reported 35% personal productivity boost** and broader job-satisfaction gains; Anthropic's 2026 report describes organizations shipping more features, fixing more bugs, and doing more "otherwise not worth it" work, with examples including **CRED doubling execution speed** and **TELUS shipping engineering code 30% faster**; Harness reports that AI-heavy teams deploy more frequently; and Faros reports improvements in throughput metrics such as epics completed and task throughput. But the best recent evidence also shows that those gains do **not** automatically translate into safer delivery, lower bug counts, or lower operational toil. [2]

For a mid-sized company, the practical answer is not "use less AI," but **"use AI inside a more disciplined system."** The methods most supported by recent evidence are: keep AI work scoped to small and easily verifiable tasks; add repository-specific context and specifications; require stricter review of AI-assisted PRs; enforce tests, static analysis, and security scanning before and during code generation; measure system-level outcomes such as review time, defect rates, deployment failures, and MTTR; and use **feature flags, progressive rollouts, approval gates, and automated rollback** to limit blast radius when AI-generated changes are wrong. Direct mid-sized-company causal evidence is still sparse, but recent case studies on controlled rollouts show feature flags are one of the most practical ways to preserve AI-driven speed without accepting a corresponding rise in operational risk. [3]

## Recent evidence window

I prioritized recent primary and near-primary sources from the six-month window above. The table below focuses on the most decision-relevant evidence for **risk, quality, efficiency, and release safety**. Where company size was not stated in the source, I mark it **unspecified**.

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026 | Industry survey | **SonarSource 2026 State of Code Developer Survey** [4] | **n=1,149** developers; org size unspecified | AI coding tools generally; specific tools not broken out | Self-reported productivity, trust, review effort, pre-commit checking | Developers reported an average **35% productivity boost**, but **96% did not fully trust AI-generated code**, **95%** spent time reviewing/ testing/ correcting it, **38%** said reviewing AI code takes more effort than reviewing coworkers' code, and only **48%** said they always check AI-assisted code before committing. [4] |
| 2026-01-28 | Sponsored industry analysis discussed on a reputable industry blog | **CodeRabbit research summarized by Stack Overflow Blog** [5] | **470 open-access GitHub repos / PRs**; company size unspecified | AI-authored PRs broadly; not one single assistant | Issues per PR, severity, logic/ correctness, security, readability | AI-authored PRs had **~1.7x more issues overall**, **1.3–1.7x more critical/major issues**, **75% more logic/ correctness errors**, **1.5–2x** more security issues, and **3x** more readability issues. [6] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026-01-29 | Academic paper | **More Code, Less Reuse** [7] | Python repositories; reviewer sentiment analysis; repository count not emphasized in snippet | Agent-generated PRs broadly | Redundancy / Type-4 clones, reviewer sentiment | AI-generated PRs showed **more semantic redundancy** than human PRs, while reviewers were **not harsher** and often more neutral or positive, implying a hidden-technical-debt risk from plausible-looking code. [8] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026-02-24 | Research update | **METR developer productivity experiment update** [9] | **57 developers, 143 repos, 800+ tasks**; smaller/open-source projects included | Agentic tools such as Claude Code and Codex; AI-allowed vs AI-disallowed conditions | Task completion time, developer/task selection effects | METR reports its earlier slowdown result is likely stale for newer tools, and newer raw estimates show some speedup, but the study design became biased because many developers refused to work without AI and selectively withheld AI-friendly tasks. That makes a strong causal estimate of current uplift difficult. [9] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026-03-21 | Academic paper | **Engineering Pitfalls in AI Coding Tools** [10] | **3,864 real-world issues** from public trackers | Claude Code, Codex CLI, Gemini CLI | Bug type, root cause, symptom, architectural layer | **67%** of observed bugs were functional; **37.3%** stemmed from API, integration, or configuration errors; the most common symptoms were API errors, terminal problems, and command failures; and bug concentration was highest in **tool/API orchestration** and **command execution** layers. [11] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026-03-30 | Academic paper | **Speed at the Cost of Quality** on Cursor adoption [12] | Matched open-source repositories; **1,380 similar control repos** in the DiD design | Cursor | Lines added, warnings, complexity | Cursor adoption was associated with **3–5x** line-added growth in the first adoption month, but those gains dissipated; meanwhile **static-analysis warnings rose 30%** and **code complexity rose 41%** post-adoption. [13] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026-03-30 / 2026-04-26 | Academic paper | **Debt Behind the AI Boom** [14] | **302,532 AI-authored commits** from **6,299 GitHub repositories** | Five assistants including GitHub Copilot and Gemini; five-tool comparison | Code smells, correctness, security, issue survival | More than **15%** of commits from every studied assistant introduced at least one issue; rates ranged from **17.4%** for GitHub Copilot to **29.1%** for Gemini; AI commits fixed more code smells than they introduced, but introduced **more correctness and security issues than they fixed**; **22.7%** of tracked AI-introduced issues still survived at the latest repository head. [15] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026-03-11 | Industry report | **Harness State of DevOps Modernization 2026** [16] | **700 engineering practitioners and managers from large enterprises** | AI coding tools generally | Deployment frequency, rollback/ hotfix/incident rate, MTTR, vulnerabilities, compliance, performance issues | Very frequent AI-coding users were more likely to deploy daily, but **69%** said AI-generated code leads to deployment problems at least half the time; **22%** of code deployments for very frequent users resulted in rollback, hotfix, or customer-impacting incident; their MTTR was **7.6 hours**; and they reported more vulnerabilities, compliance problems, performance issues, and downstream manual work. [16] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026 | Industry telemetry report | **Faros AI Acceleration Whiplash** [17] | **22,000 developers across 4,000 teams** | AI tools generally; weekly-active-tool adoption | PR size, bugs per PR, median review time, incidents per PR, code churn, throughput | Faros reports **+51% PR size, +28% bugs per PR, 5x median review time, 3x incidents per PR, and 10x code churn**, even while throughput rose. In “high AI adoption” samples, incidents per PR and bugs per developer were sharply higher. [18] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026-04-23 | Incident postmortem | **Anthropic Claude Code quality postmortem** [19] | Vendor product affecting users broadly; org size not applicable | Claude Code, Claude Agent SDK, Claude Cowork | Prompt/eval regressions, context loss, bug escape | Anthropic traced code-quality degradation to three changes: lower default reasoning effort, a context-clearing bug that made the tool “forgetful,” and a verbosity-limiting system prompt that caused a **3% eval drop**. The issues made it past human review, tests, end-to-end tests, and dogfooding, leading Anthropic to add tighter prompt controls, broader evals, and gradual rollouts. [19] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026-04 | Randomized controlled trial | **Anthropic coding-skills study** [20] | **52** mostly junior software engineers | An AI coding assistant in a coding environment; not agentic Claude Code | Task time, quiz mastery, debugging/ code-reading/ conceptual understanding | AI assistance made participants finish only **slightly faster** and **not significantly so**, but the AI group scored **17% lower** on mastery and had the biggest weakness on debugging questions. Heavy delegation patterns were associated with the worst scores. [21] |

| Date | Study type | Source | Sample or company size | AI tools involved | Metrics used | Main finding |
|---|---|---|---|---|---|---|
| 2026 | Official report with case studies | **Anthropic 2026 Agentic Coding Trends Report** [22] | Internal studies plus named company cases; mostly large organizations; mid-size often unspecified | Claude Code / Claude agents | Time saved, speed, output volume | Anthropic reports engineers use AI in roughly **60%** of their work but fully delegate only a small fraction; about **27%** of AI-assisted work is work that otherwise would not have been done; **CRED** doubled execution speed while keeping humans in the loop; **TELUS** shipped engineering code **30% faster** and reported **500,000+ hours saved**. [23] |

Two points stand out from this evidence base. First, **the strongest recent quality-risk evidence is not just survey sentiment**: it now includes large repository mining studies, issue-tracker studies, and enterprise telemetry. Second, **mid-sized-company-specific causal evidence remains weak**. Most recent primary data comes from open-source repos, large enterprises, or vendor product telemetry, so any direct claim about "mid-sized companies specifically" should be treated as an extrapolation rather than a settled empirical result. [24]

## Root causes and risk mechanisms

The recent evidence does **not** support a simplistic story that hallucination alone is the main problem. Instead, the dominant recent pattern is that AI increases errors through a mix of **plausible-but-wrong logic, orchestration/integration failures, context loss, weak requirement coverage, and human-review overload**. Academic work on real issue trackers found most bugs in AI coding tools arise from **API/**

**integration mismatches, configuration/setup errors, environment incompatibilities, and tool-orchestration failures**, not only from model reasoning failures. That matters operationally: if a company focuses only on prompt quality and ignores the surrounding toolchain, it will miss a large part of the failure surface. [11]

Sonar's survey and Anthropic's April postmortem help explain why these bugs escape. Developers report that AI often produces code that **looks correct but is not reliable**, and Anthropic documented a real case where prompt and context-management changes degraded coding quality even though the changes passed multiple reviews and automated tests. In other words, the modern failure mode is often **"quietly wrong and easy to merge,"** not "obviously broken." [25]

A second mechanism is **review saturation**. Faros reports much larger PRs and far longer median review time under heavy AI adoption; GitHub's May 2026 guidance explicitly warns that agent-generated PR volume is already saturating review bandwidth, that agents often duplicate existing utilities, and that reviewers must trace one critical path rather than scanning the diff superficially. This aligns with the "More Code, Less Reuse" paper, where AI-generated PRs were more redundant but were not punished by reviewers. [26]

A third mechanism is **skill and ownership erosion**, especially for junior developers or when the task requires new conceptual understanding. Anthropic's randomized trial found that AI users did not gain statistically significant speed but did lose mastery, especially on debugging, and the BNY Mellon mixed-method study argues that productivity measurement should include long-term dimensions such as technical expertise and ownership of work, not just short-term throughput. The maintenance study published in May 2026 adds a further nuance: even when AI-generated files are not obviously burdening teams with immediate corrective maintenance, **humans still performed about 83% of the maintenance** on those AI-generated files, suggesting that the work often shifts rather than disappears. [27]

| Root cause or mechanism | Recent evidence | Typical failure mode | Best-supported mitigations |
|---|---|---|---|
| Plausible but incorrect logic | Sonar; CodeRabbit; GitHub review guidance [28] | Off-by-one errors, missing permission checks, bad branching, incorrect dependency flow | Critical-path tracing in review, required failing regression tests, canary rollout, feature flag kill switch |
| API / integration / configuration mismatches | Engineering Pitfalls study [11] | Broken commands, wrong tool invocation, environment-specific failures | Standardized dev environments, pinned tool versions, integration tests, authoring-stage static checks |
| Context loss, memory bugs, and prompt drift | Anthropic postmortem [19] | Forgetfulness, repetitive edits, degraded code quality after prompt changes | Broader eval suites, soak periods, prompt-change governance, gradual rollouts, exact-public-build dogfooding |

| Root cause or mechanism | Recent evidence | Typical failure mode | Best-supported mitigations |
|---|---|---|---|
| Redundancy, verbosity, and code reuse blindness | More Code, Less Reuse; GitHub review guidance; Faros telemetry [29] | Duplicate helpers, unnecessarily large diffs, more churn and review time | Require small PRs, search for existing utilities before merge, automated review instructions for reuse checks |
| Incomplete understanding and requirement coverage | Anthropic skill trial; recent maintenance study; BNY Mellon study [30] | AI handles the first draft, humans must later extend, explain, or debug it | Ask AI for explanations, not only code; restrict autonomous use on unfamiliar domains; track downstream maintenance and ownership |

## Methods that improve efficiency without increasing risk

The most practical conclusion from the recent evidence is that companies should **optimize for "AI with guardrails," not "AI instead of guardrails."** The best-supported methods below preserve the parts of AI that create value—faster drafting, boilerplate generation, local cleanup, quicker experimentation—while constraining the failure modes that drive bugs, incidents, and technical debt. [31]

| Method | Why it improves efficiency | Why it lowers risk | Evidence strength |
|---|---|---|---|
| Keep AI work small, scoped, and easily verifiable | Small tasks and smaller PRs are faster to generate and easier to review; AI is especially useful for localized changes and repetitive work. [32] | Smaller diffs reduce hidden logic errors, duplication, and reviewer overload; GitHub explicitly recommends rejecting oversized agent PRs without a clear implementation plan. [33] | High |
| Add repository-specific context and specifications | GitHub's context engineering guidance shows custom instructions, reusable prompts, and custom agents reduce back-and-forth prompting and increase consistency. [34] | Better context reduces architecture drift and misuse of conventions; Anthropic's postmortem shows that prompt and context management materially affect quality. [19] | Medium |
| Require critical-path human review for AI-assisted logic changes | AI handles low-level scanning and drafting; humans spend scarce time on the most important path through the change. [33] | This directly targets the "looks right but is wrong" problem and the review-bandwidth problem. GitHub recommends checking CI weakening first, tracing a critical path end-to-end, and requiring a pre-fix failing test. [33] | High |

| Method | Why it improves efficiency | Why it lowers risk | Evidence strength |
|---|---|---|---|
| Move quality checks earlier than PR merge | Static analysis, security checks, and authoring-stage scanning catch defects before large AI diffs accumulate review/merge momentum. [35] | Recent academic evidence recommends normalizing static analysis, tests, and security checks around AI-generated code because correctness and security issues survive longer than code smells. [36] | High |
| Use AI for **comprehension**, not just delegation | Anthropic's trial found the best learning patterns were asking for explanations and conceptual help, not handing over the whole task. [37] | This reduces skill decay and improves future debugging capacity, which the BNY Mellon study identifies as a neglected but important long-term productivity dimension. [38] | High |
| Measure end-to-end outcomes, not only coding speed | Throughput metrics capture real gains, but they must be paired with DORA-style stability metrics, review time, churn, and defect outcomes. [39] | This prevents a false positive where AI "improves productivity" only by shifting work downstream into review, QA, incidents, or rework. [40] | High |

A concrete operating model for a mid-sized engineering organization is:

1. **Allow AI by default** for scaffolding, test generation, refactoring, docs, and simple bug fixes.
2. **Require human-authored specs or explicit acceptance criteria** before AI touches multi-file business logic or security-sensitive code.
3. **Block merge** on unit tests, static analysis, secret scanning, and policy checks.
4. **Flag AI-assisted PRs** so reviewers know to inspect critical paths, duplication, CI changes, and permission boundary changes more carefully.
5. **Deploy dark behind server-side flags**, then release progressively using canary cohorts and automated rollback on SLO regressions.
6. **Track post-merge outcomes** such as bug rate, review time, deployment failures, MTTR, code churn, and the percentage of AI-generated code later rewritten by humans. This is the closest thing to a safe "AI ROI dashboard" supported by recent evidence. [41]

Two recent case snippets are especially instructive. Anthropic's April 2026 postmortem shows that even AI vendors themselves can ship coding-quality regressions that survive review and tests; their response—**broader eval suites, tighter prompt-change controls, additional context for code review, and gradual rollouts**—is exactly the kind of governance pattern companies should mirror. GitHub's May 2026 review guidance is similarly concrete: **treat CI weakening as a hard stop, let automated review catch the mechanical issues first, and force reviewers to trace one critical path and demand a failing test for any non-trivial logic change**. [42]

## Feature flags, gates, and rollback design

Feature flags are not a substitute for code review, testing, or static analysis. They do **not** prevent hallucinated APIs, bad logic, or insecure code from being written. What they do is **reduce blast radius after merge**, **separate deployment from release**, **enable progressive exposure**, and **give teams a fast kill switch** when AI-generated changes misbehave in production. In the recent evidence, this is one of the clearest practical mechanisms for preserving AI-driven throughput without accepting the full operational downside. [43]

Recent Harness research is useful here because it ties AI coding directly to release risk. In its 2026 "State of AI in Software Engineering" report, Harness found feature-management/rollout automation lagging behind coding automation, and showed that shipping-frequency gains from AI were much stronger in organizations with more mature CD automation. In its 2026 DevOps modernization study, Harness concluded that teams need standardized, governed pipelines including **feature flags and automated rollbacks** to keep faster AI-generated change from overwhelming delivery systems. [44]

The company case-study evidence is not AI-specific, but it is highly relevant because it addresses the exact operational risks that AI coding magnifies: too much change volume, too many risky releases, and too much human remediation.

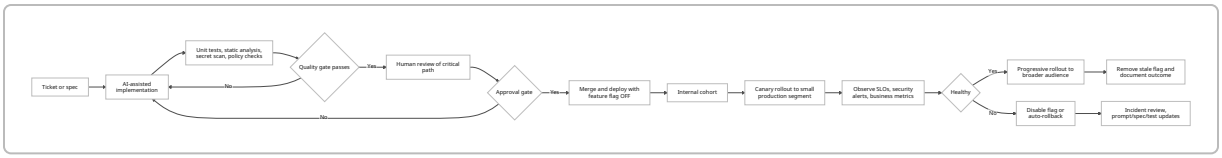
| Source | Company size or scope | Flag or gate mechanism | Reported outcome | What it means for AI-assisted coding |
|---|---|---|---|---|
| **Harness State of AI in Software Engineering** [45] | **900 engineers, platform leaders, and managers** across regions | CD automation, feature management/ rollouts, deployment verification | AI-heavy orgs ship faster, but only **6%** say CD is fully automated; organizations with fully automated pipelines were far more likely to convert AI coding into more frequent shipping. [45] | AI speed converts into real delivery benefit only when downstream controls mature, including rollout control. |
| **Harness State of DevOps Modernization 2026** [16] | **700** large-enterprise respondents | Standardized, governed pipelines; explicit recommendation for **feature flags** and **automated rollbacks** | Very frequent AI users had **22%** remediation/ incident/rollback-like outcomes and **7.6h MTTR**; the report recommends feature flags and automated rollback as part of the remedy. [16] | Flags address the operational half of the AI problem: bad code that escapes into production. |

| Source | Company size or scope | Flag or gate mechanism | Reported outcome | What it means for AI-assisted coding |
|---|---|---|---|---|
| **Swedbank case study** [46] | **50+ teams**, large bank | Feature flags plus canary-style real-time testing | More frequent releases, reduced release risk, independent frontend/backend/mobile coordination, safer testing in production. [46] | Especially relevant where AI increases change volume across multiple services and teams. |
| **Vida Health case study** [47] | Company size unspecified | Feature flags, phased rollouts, production testing, instant rollback | Mobile release cadence improved from **monthly to weekly**; **30%** more features per release; releases stabilized by phased rollouts and rollback capability. [47] | A good pattern for mid-sized product teams: push code often, expose slowly, watch metrics closely. |
| **Experian case study** [48] | **3,000** product + engineering employees | Controlled rollouts behind flags | Moved from **2** big-bang releases per month to **100** risk-controlled deployments per month. [48] | Flags can absorb the deployment-frequency shock that AI coding creates. |
| **GitHub review guidance** [33] | General engineering practice | Human approval gate, least-privilege workflow permissions, separation of analysis from execution | GitHub recommends a human approval gate for anything that can touch production, no evaluation of raw model output as shell commands, and strict permission scoping. [33] | "Gate" design matters just as much as "flag" design when AI is active inside CI/CD. |

A sensible **minimum viable rollout pattern** for a mid-sized company is:

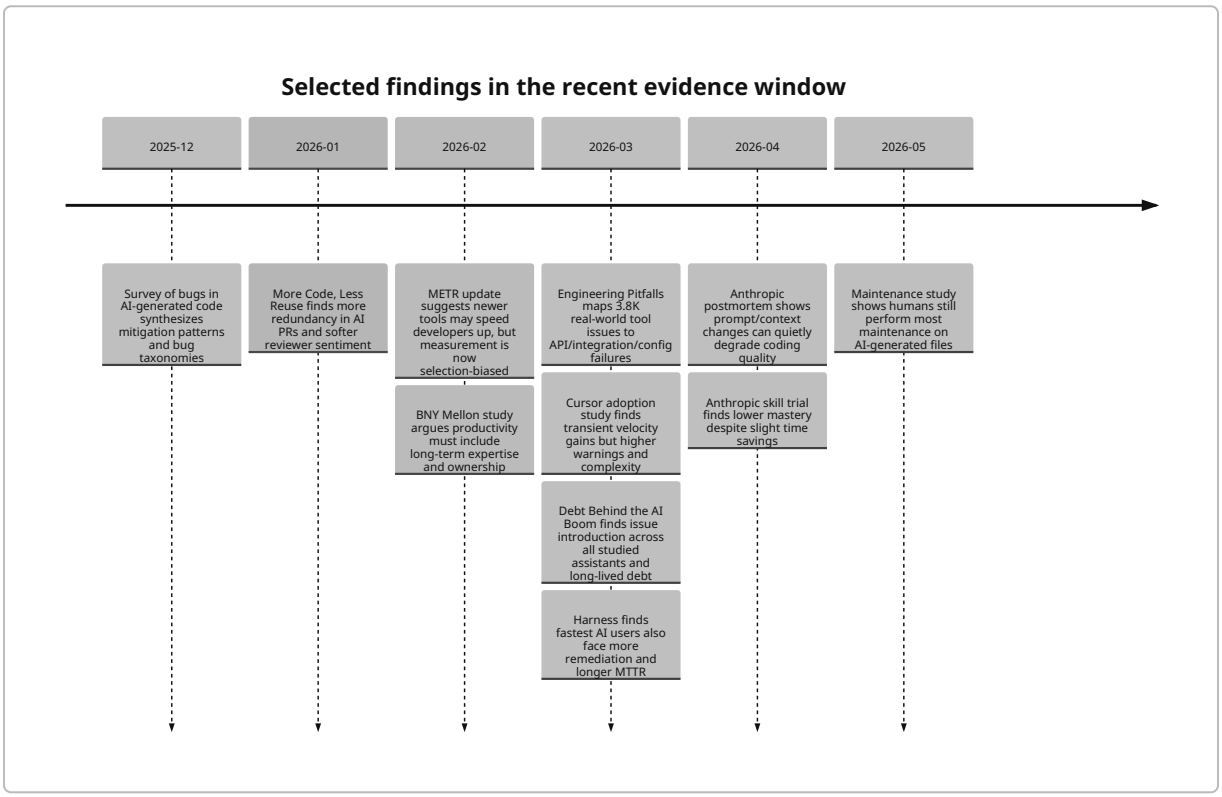
- **Deploy with the feature off** by default.
- **Turn on for internal users first**.
- **Canary to a small customer cohort**.

- **Watch SLOs plus business and security signals**.
- **Auto-disable or roll back** on error-rate, latency, or incident-threshold breach.
- **Remove the flag** after the rollout stabilizes so flags do not become another form of technical debt. [49]

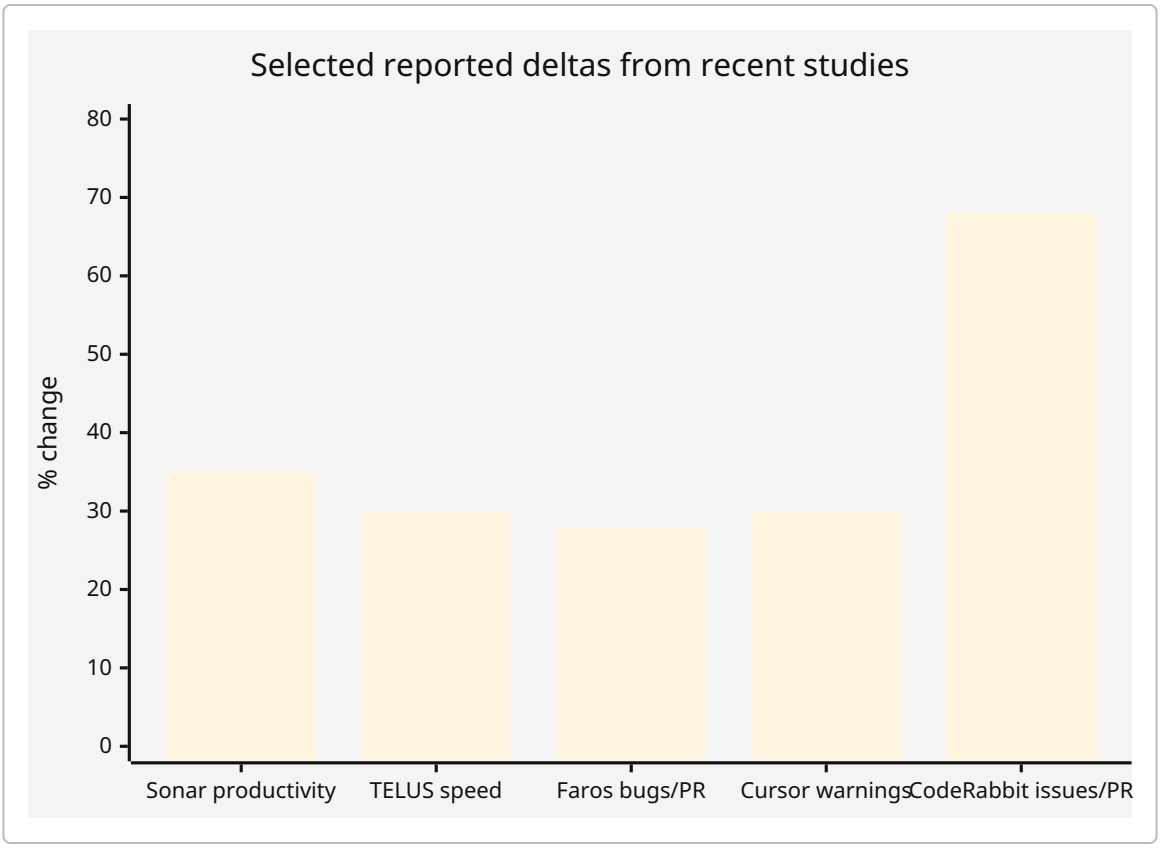



## Visualizations and limitations

The timeline below highlights how quickly the recent evidence coalesced around the same pattern: **more code, more speed, and more need for control**. The dates are publication dates or report dates within the requested window. [50]




The chart below is **directional, not apples-to-apples**: it compares percentage deltas from different methodologies and different metrics. It is useful for seeing the shape of the evidence, not for claiming that all bars measure the same thing. The first two bars are efficiency gains; the last three are risk increases. [51]




The main limitations are straightforward. **Direct mid-sized-company causal evidence is sparse**; most recent hard data is from open-source repositories, vendor telemetry, or large-enterprise surveys. Several sources are also vendor-produced, which makes them useful for operational patterns and metrics but weaker than peer-reviewed causal studies for estimating true average effects. Finally, the studies use heterogeneous definitions of "AI-generated code," different tool mixes, and different outcome metrics, so the safest high-confidence conclusion is qualitative: **AI-assistance often increases engineering throughput, but unless review, testing, rollout, and rollback systems are upgraded too, it also tends to increase bugs, incidents, review effort, and long-term maintenance risk.** [52]

---

[1] [17] [18] [26] [31] [52] https://www.faros.ai/research/ai-acceleration-whiplash
https://www.faros.ai/research/ai-acceleration-whiplash

[2] [4] [25] [28] [51] https://www.sonarsource.com/state-of-code-developer-survey-report.pdf
https://www.sonarsource.com/state-of-code-developer-survey-report.pdf

[3] [34] https://github.blog/ai-and-ml/generative-ai/want-better-ai-outputs-try-context-engineering/
https://github.blog/ai-and-ml/generative-ai/want-better-ai-outputs-try-context-engineering/

[5] https://stackoverflow.blog/2026/01/28/are-bugs-and-incidents-inevitable-with-ai-coding-agents/
https://stackoverflow.blog/2026/01/28/are-bugs-and-incidents-inevitable-with-ai-coding-agents/

[6] https://coderabbit.ai/blog/state-of-ai-vs-human-code-generation-report
https://coderabbit.ai/blog/state-of-ai-vs-human-code-generation-report

7 29 https://arxiv.org/html/2601.21276v1
https://arxiv.org/html/2601.21276v1

8 https://arxiv.org/abs/2601.21276
https://arxiv.org/abs/2601.21276

9 https://metr.org/blog/2026-02-24-uplift-update/
https://metr.org/blog/2026-02-24-uplift-update/

10 11 https://arxiv.org/html/2603.20847
https://arxiv.org/html/2603.20847

12 13 https://cmustrudel.github.io/papers/msr2026he.pdf
https://cmustrudel.github.io/papers/msr2026he.pdf

14 15 24 35 36 https://arxiv.org/html/2603.28592v2
https://arxiv.org/html/2603.28592v2

16 40 https://www.harness.io/state-of-devops-modernization-2026
https://www.harness.io/state-of-devops-modernization-2026

19 42 https://www.anthropic.com/engineering/april-23-postmortem
https://www.anthropic.com/engineering/april-23-postmortem

20 21 27 30 37 https://www.anthropic.com/research/AI-assistance-coding-skills
https://www.anthropic.com/research/AI-assistance-coding-skills

22 https://resources.anthropic.com/2026-agentic-coding-trends-report
https://resources.anthropic.com/2026-agentic-coding-trends-report

23 https://resources.anthropic.com/hubfs/2026%20Agentic%20Coding%20Trends%20Report.pdf
https://resources.anthropic.com/hubfs/2026%20Agentic%20Coding%20Trends%20Report.pdf

32 https://arxiv.org/html/2603.27130v2
https://arxiv.org/html/2603.27130v2

33 41 https://github.blog/ai-and-ml/generative-ai/agent-pull-requests-are-everywhere-heres-how-to-review-them/
https://github.blog/ai-and-ml/generative-ai/agent-pull-requests-are-everywhere-heres-how-to-review-them/

38 39 https://arxiv.org/html/2602.03593v1
https://arxiv.org/html/2602.03593v1

43 49 https://www.harness.io/blog/how-to-release-confidently-at-ai-speed
https://www.harness.io/blog/how-to-release-confidently-at-ai-speed

44 45 https://www.harness.io/the-state-of-ai-in-software-engineering
https://www.harness.io/the-state-of-ai-in-software-engineering

46 https://www.harness.io/case-studies/swedbank-speeds-up-software-releases-with-feature-flags
https://www.harness.io/case-studies/swedbank-speeds-up-software-releases-with-feature-flags

47 https://www.harness.io/case-studies/vida-health-accelerates-product-delivery-and-lowers-risk-with-feature-flags
https://www.harness.io/case-studies/vida-health-accelerates-product-delivery-and-lowers-risk-with-feature-flags

48 https://www.harness.io/case-studies/how-experian-increased-release-velocity-by-50x
https://www.harness.io/case-studies/how-experian-increased-release-velocity-by-50x

50 https://arxiv.org/abs/2512.05239
https://arxiv.org/abs/2512.05239