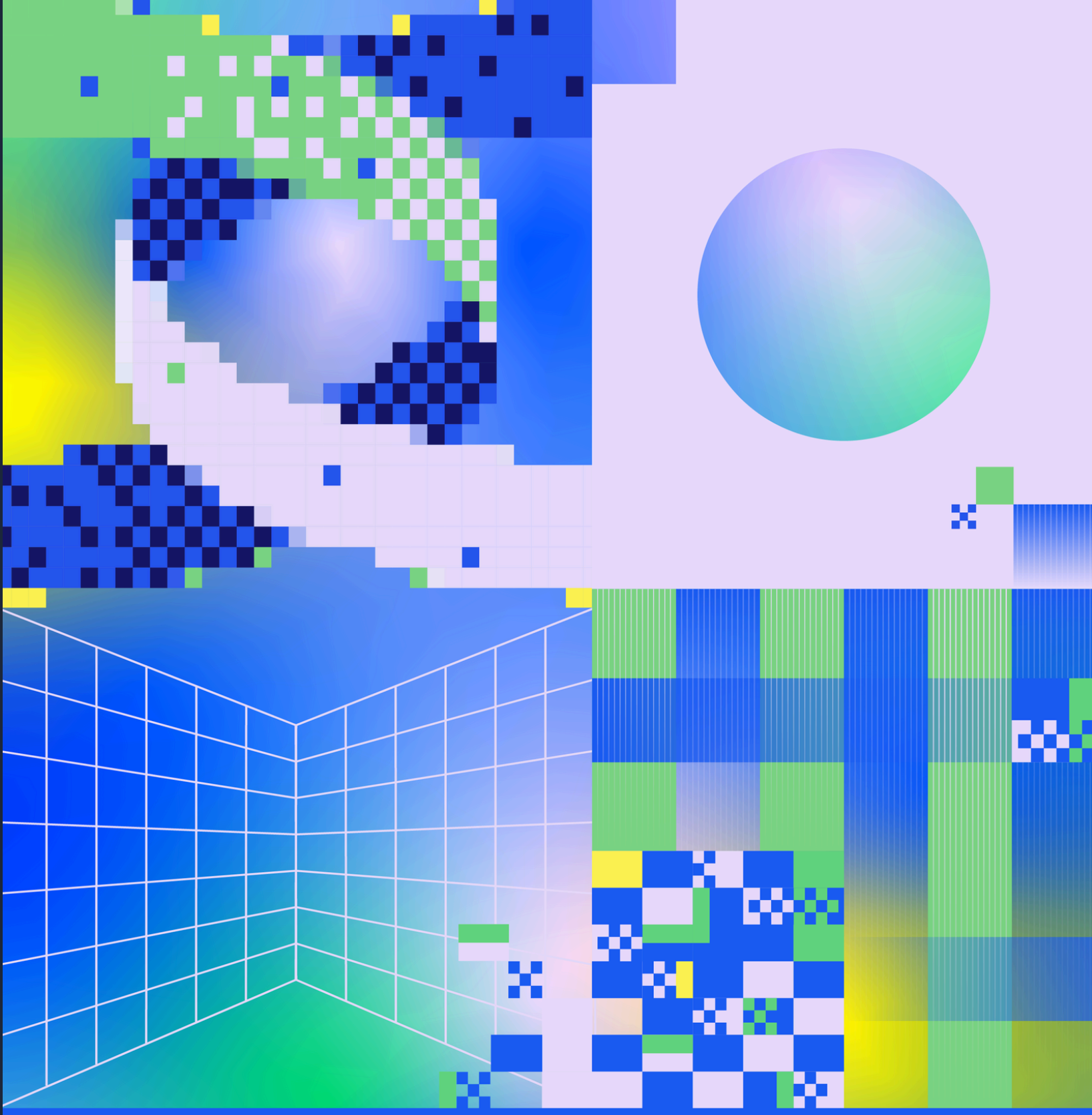


2026 | Q2 Pulse Report

State of Software Delivery

circleci

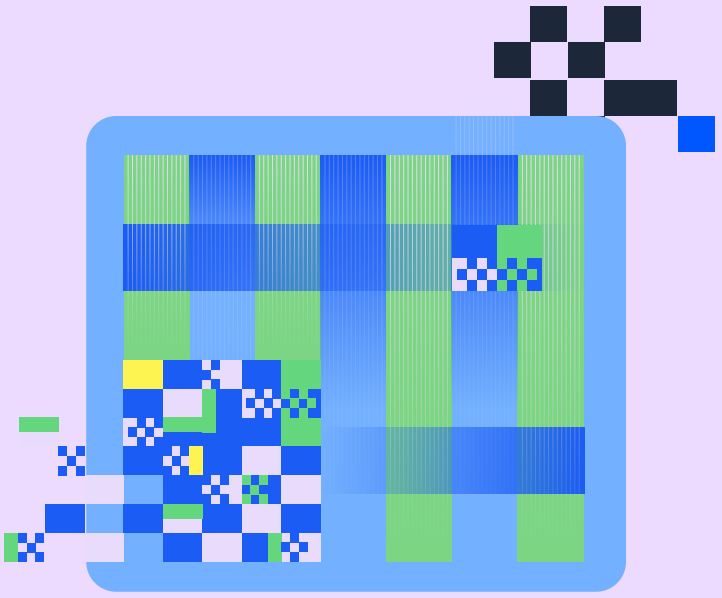
// Contents

// Foreword

FOR THE PAST SIX YEARS, the State of Software Delivery has tracked how software gets built, validated, and shipped using one of the largest datasets of real software delivery metrics in the world. Engineering organizations have relied on these benchmarks to navigate massive technological and methodological changes, from the rise of cloud computing to the emergence of platform engineering.

AI represents a different kind of shift. The pace of change today is faster than anything we have seen before. New models, tools, and capabilities are reaching the market every day, and engineering leaders are making decisions now that will shape their delivery systems and cost structures for years to come. The need for up-to-the-minute data has never been greater.

That's why we're following up on our annual State of Software Delivery report, published every Q1, with this Pulse report. Pulse reports are shorter, more frequent check-ins between annual editions, built to surface what's changing while the data is still fresh. Our goal is to provide these timely insights so you can adjust your strategy as fast as the technology itself is moving.



// Executive summary

The 2026 State of Software Delivery identified a growing disconnect at the center of modern software development: teams are producing more code but shipping less of it. Feature activity accelerated while validated delivery slowed, revealing a widening delivery bottleneck.
Only a small subset of organizations successfully converted increased development volume into production throughput. In this report, we dig into those outliers.

Who are the teams that are actually shipping more? What operational behaviors separate them from the rest of the market? And, importantly, what does it cost to perform at that level?

Every engineering leader is facing the same dual mandate: **HOW DO WE MOVE FASTER WITHOUT LOSING CONTROL OF COST?** This report shows that the achievement of both those goals is driven by the same underlying mechanics.
The practices that allow elite teams to break through delivery constraints also reshape the economics of software development, from CI infrastructure spend to AI token consumption.

To understand how leaders are pulling ahead, we analyze both the highest-performing organizations in our dataset and the operating metrics behind their results. We focus particularly on a new metric: Merge Efficiency Ratio (MER), the number of validation cycles required to move a change from development into production, which acts as both a performance metric and a cost metric in the age of agentic development.

Key findings

- **The delivery bottleneck remains the defining challenge of the AI era:** Feature-branch activity continues to grow, but workflow activity on production branches remains sluggish. Teams continue to generate more code than they can efficiently ship.
- **A small breakout cohort is widening the performance gap:** The top 5% of organizations now ship validated code 9x faster than the median team, up from 8x in Q1.
- **The teams pulling ahead are improving merge efficiency:** Elite performers run far fewer validation cycles per shipped change, lowering MER from a population median of 3.9 to 1.3. That efficiency partly explains why they reduced cost per shipped change 31% while median teams saw it rise 13%.
- **Moving validation into the inner loop is the path forward for every team:** Catching routine failures before code reaches CI keeps feedback fast and contexts warm, which is how the leaders ship more at lower cost. For a 50-person team, giving faster feedback earlier in the delivery cycle can cut total delivery costs, including both AI and CI spend, by $700K or more per year. This pattern is available to any team, regardless of size or budget.

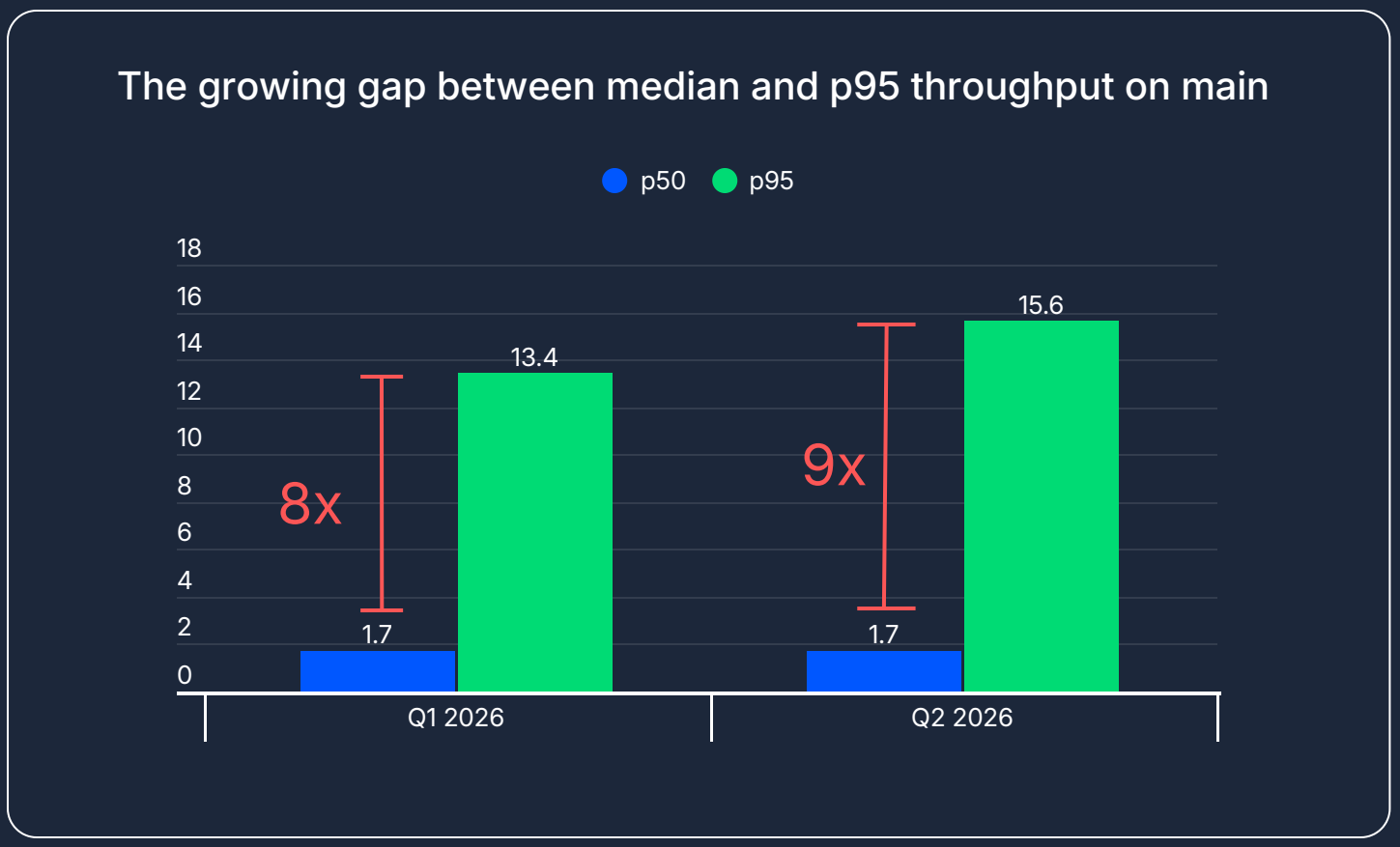
In the AI era, software organizations do not win by producing more code. They win by building systems that turn more of that code into production outcomes at sustainable cost. In the pages that follow, we'll give you the roadmap to do just that.

//The velocity gap is widening

Top teams ship 9x the median

Perhaps the most important finding of the Q1 State of Software Delivery is that AI is pushing code volume up almost everywhere, but the productivity gains aren't landing evenly across organizations.

We're seeing this divergence unfold at an accelerating pace. In Q1, main-branch throughput (the number of workflows run on the default branch, indicating production-bound code) at the 95th percentile was eight times the median. In other words, across all users, the top 5% shipped around 8x the code per day as the typical team. In Q2, it hit **9x.**



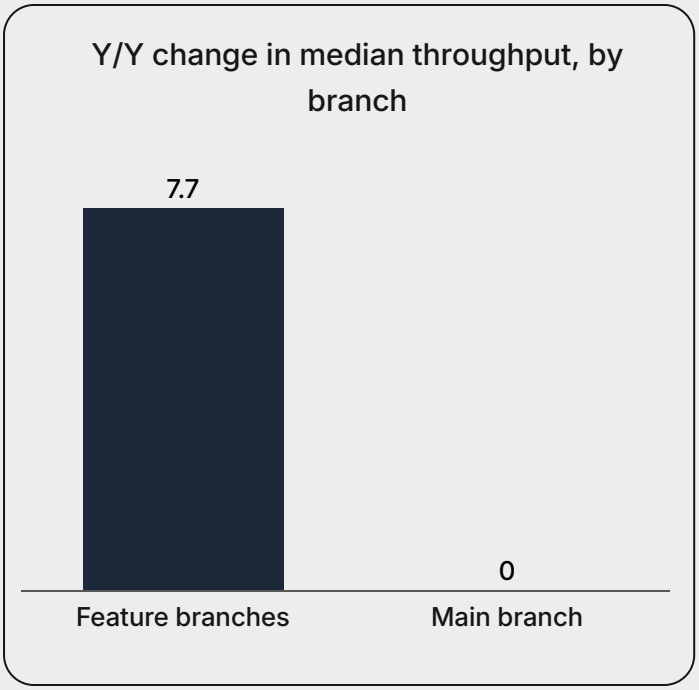
The teams at the top keep moving more validated code into production, while the middle treads water. Everyone is writing more code than before; what separates them (and the only measure that matters) is how much of it actually reaches a production branch.

02 Validation is still the industry's biggest bottleneck

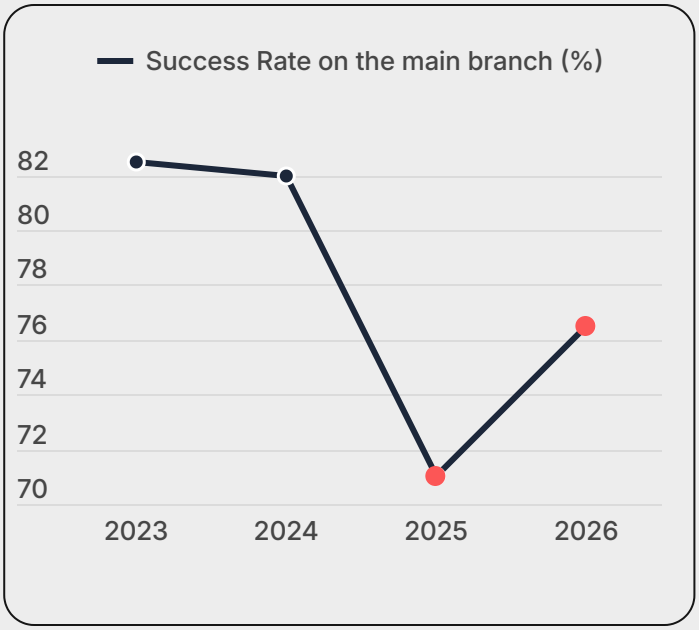
For most teams, the primary obstacle to moving AI-generated code into production is their ability to safely integrate it. Large, untrusted merge requests create massive review backlogs and complex failures that prevent code from reaching main as efficiently as it's produced.

The metrics reveal how this creates a validation bottleneck for most teams:

- **Feature-branch throughput** (the number of workflows run on isolated development branches) grew **7.7% year over year.**
- **Main-branch throughput** (the number of workflows run on primary integration branch) **stayed entirely flat.**



The Q1 State of Software Delivery found that main-branch throughput had declined year over year, so a flat annual growth rate in Q2 represents a modest improvement. One reason for this change in trajectory is that success rates on main ticked up to **76.7**% in Q2 from **70.8%** in Q1. Fewer failures on main mean less churn preventing the next change from moving through the pipeline.



Even so, this baseline remains well below the mid-80s results of 2023-2024 and even further below our recommended 90% success rate benchmark. Validating code is the core bottleneck now, and the typical team continues to struggle.

03 The breakout cohort

Percentiles are useful for illustrating the size of the gap between typical and top performers. But they don't show what the fastest teams are actually doing differently. For that, we looked at an anonymized group of 20 real organizations with the highest main-branch throughput on CircleCI.

Where the typical main-branch workflow runs about 1.7 times a day, and the busiest 5% run about 15.6 times per day, these 20 elite organizations average roughly 2,165 workflows a day on the main branch. They are rapidly pulling away from the pack, too: that organizational average has surged 72% over the past year.

	p50	p95	Top 20
Daily throughput on main	1.71	15.63	2,165
Year-over-year change	Flat	+13.4%	+72%

While this isn't an exact comparison to the percentile figures above (those measure individual workflows, while this counts every main-branch workflow an organization runs), these 20 teams show what it looks like to turn higher code volume into shipped software, the thing most teams are still struggling to do.

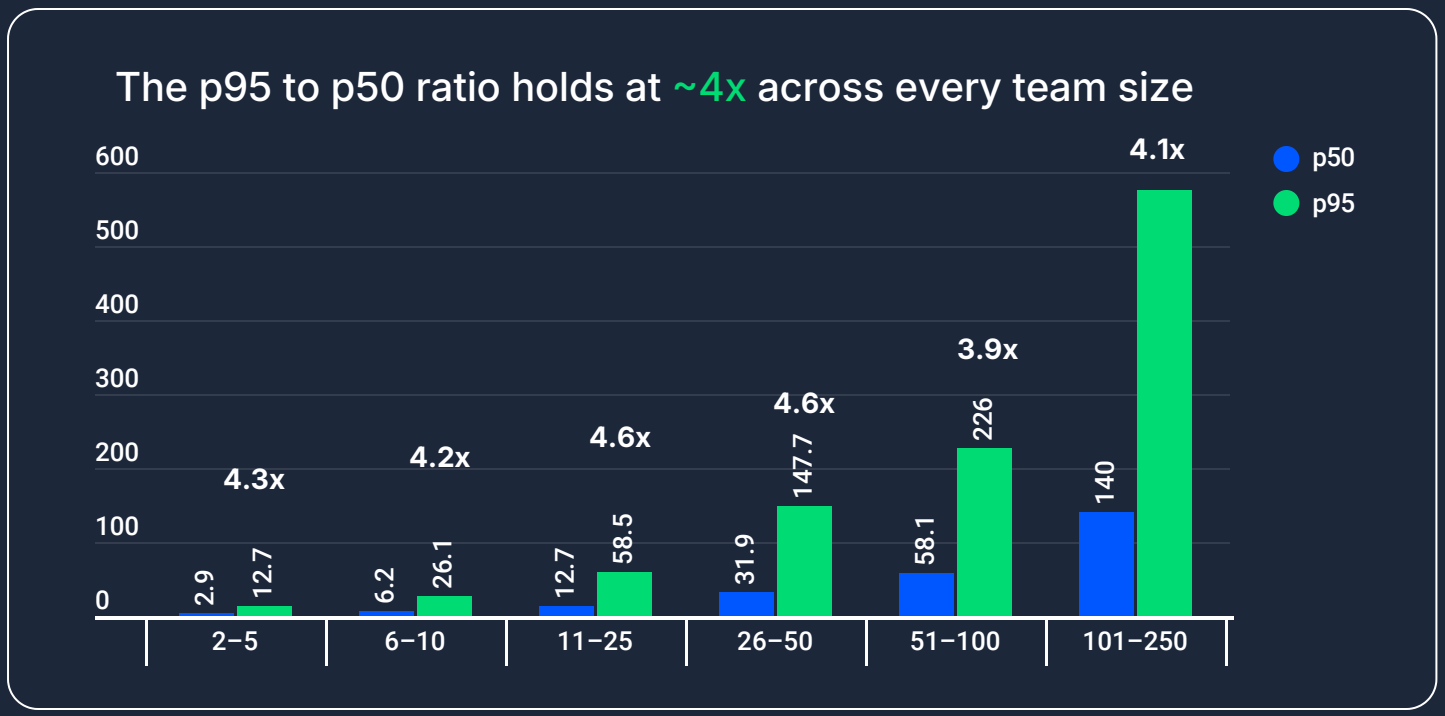
This cohort spans a range of regions, sizes and industries:

- **Region:** 50% North America, 35% EMEA, 10% Asia-Pacific, with about half headquartered in the United States.
- **Size:** Mid-to-large engineering organizations, ranging from roughly 20 to 1,400 active contributors, with a median near 290.
- **Industry:** Clustered in security and developer tooling, with fintech, e-commerce, logistics, and energy also represented.

04 Debunking the scale myth

One of the most common questions we hear about top performers is whether their edge just comes down to size. It's a fair assumption. Bigger orgs have more engineers, more budget, more infrastructure, and they do ship more code in absolute terms. So it would be easy to write off elite throughput as something only large organizations can reach.

The data says otherwise. When we bucket teams by headcount, the gap between top performers (p95) and typical teams (p50) stays remarkably steady at every size, holding at roughly 4x whether a team has 5 engineers or several hundred.



That consistency means more than "there are high performers at every size." It means smaller teams are regularly out-shipping much larger ones. A top-performing team of 2 to 5 engineers ships as much validated code per day as the median team of 11 to 25. The best teams in the 11 to 25 range keep pace with the typical org of 51 to 100. A top mid-sized team of 26 to 50 out-ships the median organization of 101 to 250 outright.

So the advantage clearly isn't about size. It's about how these teams work. The sections that follow break down what high performers do differently, starting with the most basic measure of all: how much each individual engineer is able to ship.

//Habits of high-performing teams

The data so far shows that high throughput isn't just a function of company size or engineering headcount. The teams moving fastest are moving work more efficiently through the path from development to production.

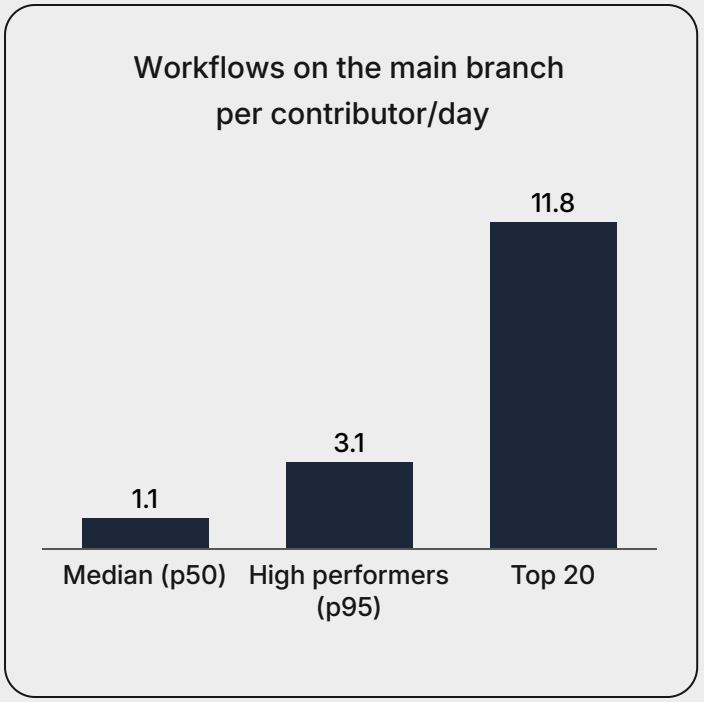
This chapter looks at the measurable behaviors behind that difference. High performers use CI as a continuous validation layer beyond ordinary code pushes, spend fewer cycles retrying work on feature branches, and create delivery systems where each contributor can get more finished work across the line.

Each section connects one of those signals to a practical habit teams can start building into their own workflow.

05 Habit one: High individual output

The fastest teams get more production-ready work out of each engineer. This happens because their systems make it easy to get code safely into production without long waits, large review queues, or repeated rounds of late-stage cleanup. Across the dataset, the difference is clear:

- **Median (p50):** typical developer averages just 1 completed main-branch update per day.
- **High Performers (p95):** Individual contributors safely merge 3 updates to main every single day.
- **Top 20 Teams:** Every contributor averages 12 successful main-branch updates per day.



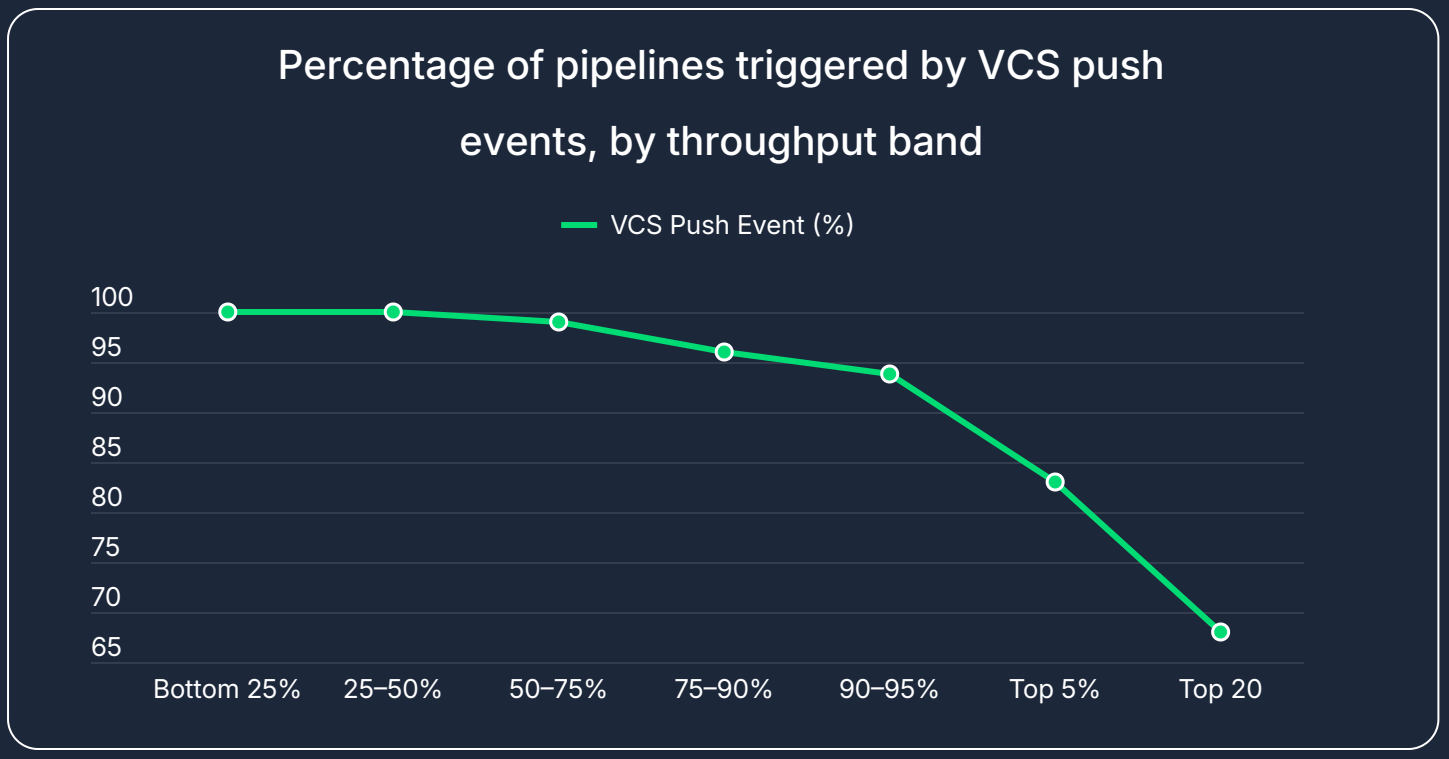
The Top 20 are also improving this number quickly. Last year, they averaged 6.5 main-branch workflows per contributor per day. This year, they averaged 12, nearly doubling the amount of validated work reaching main per engineer.

One way teams create that kind of flow is by keeping work close to main. In a trunk-based model, developers work from the main branch, keep branches short-lived, and integrate continuously instead of letting changes drift for days or weeks before review. That matters even more with AI-assisted work: the larger and longer-lived a generated change becomes, the harder it is to review, validate, and merge cleanly.

Where to start: Keep AI-generated code focused on discrete changes, and validate early as the work develops. Start with a short implementation plan, make one bounded change, then run the smallest useful validation pass: the tests, linting, type checks, or build steps relevant to that change. Feed any failures back into the next iteration until the relevant checks pass, then open the PR while the diff is still small enough to review, validate, and ship with confidence.

06 Habit two: Continuous validation

High-throughput teams use CI for more than just responding to code pushes. As teams move up the performance tiers, their workflows are triggered by a larger share of automated and scheduled validation alongside ordinary commit-driven runs.



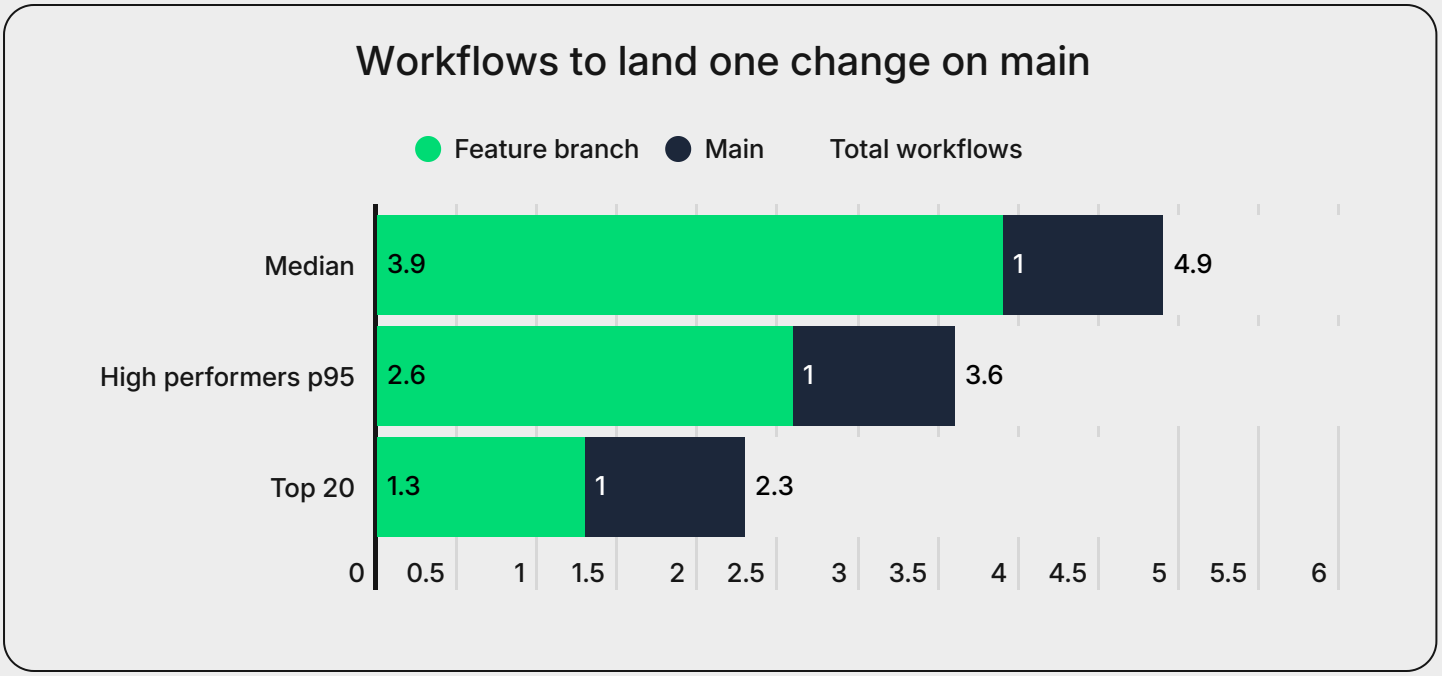
The median team triggers nearly 100% of its CI runs from direct repository pushes. For p95 teams, commit-driven pushes account for 83% of pipeline triggers. For the Top 20, that share drops to 68%, meaning roughly a third of their pipeline activity comes from non-commit triggers: scheduled workflows, API-triggered workflows, dependency-update workflows, and post-deploy verification.

A broader trigger mix gives teams more places to run validation across the delivery lifecycle, from dependency health and compliance checks to release verification and post-deploy smoke tests. Actively scanning for these issues outside of the commit path keeps push-driven feedback loops focused on the change itself and prevents unrelated failures from blocking deployments.

Where to start: Look at the workflows that run on every code push. Split them into two groups: the checks that genuinely need to block a merge, and the ones that only need to run on a regular basis. Keep the blocking checks on your commit-driven pipeline so feedback stays focused on merge readiness. Move the rest into separate scheduled, API-triggered, or webhook-driven workflows.

07 Habit three: Low merge efficiency ratio (MER)

High-throughput teams spend fewer cycles getting code from feature branches to main. We track this with Merge Efficiency Ratio, or MER: the number of feature-branch workflow runs for every workflow run that lands on main. MER shows how much validation work happens before a change reaches the main branch. At the median, MER is 3.9, which works out to roughly 5 total CI cycles when you include the final run on main. At p95, MER drops to 2.6. For the Top 20, it drops to 1.3.



Lower MER means less time spent cycling on isolated branches before work becomes production-ready. This is where practices like trunk-based delivery matter. Shorter-lived branches, earlier validation, and more frequent integration all reduce the number of round trips it takes to land a change. In AI-assisted workflows, each round trip carries a potential double cost: the tokens spent re-analyzing the same code and regenerating fixes, and the merge conflicts that pile up as the rest of the team keeps shipping. The more often that loop repeats, the more costly a bottleneck validation becomes.

Where to start: Look at how many times a typical branch runs CI before it merges. If branches need several validation passes before they land, find the repeat work: missing tests, unclear AI-generated changes, stale branches, environment issues, or checks that could run earlier. Pick the most common source of extra cycles and move that feedback closer to where the change is created. The goal is to reduce the number of round trips from first validation to merge.

08 Merge efficiency is improving fastest at the top

The teams with the strongest throughput are not standing still on merge efficiency. They already spend fewer validation cycles getting work to main, and they continue to reduce that overhead. Whether this is a deliberate shift or a byproduct of other priorities like increasing team velocity or eliminating waste, the pattern is clear.

MER Performance Year-over-Year

Cohort	Q2 2025 MER	Q2 2026 MER	YoY Change
Median	3.98	3.91	-1.7%
Top 20 Orgs	1.62	1.28	**-21%**

For the median team, MER barely moved year over year. For the Top 20 organizations, MER improved from 1.62 to 1.28, a 21% reduction in feature-branch validation cycles per main-branch run.

MER is becoming more than a static performance benchmark. It is one of the places where high-throughput teams continue to separate from the rest of the field. They are moving more work through CI while reducing the amount of validation effort spent away from main.
This has implications not just for speed, but for cost as well.

MER is a developer experience metric, too

So far, we've treated MER as a signal of delivery efficiency: how much validation work it takes to get code from a feature branch to main. But MER also has a direct effect on another metric high-performing teams care deeply about: developer experience.

When MER is high, developers feel it in their daily work. They spend more time waiting for checks, switching back into work they thought was done, and figuring out whether a problem came from their change, a stale branch, or something else moving in the codebase. The longer that cycle runs, the less of their effort turns into shipped work.

Improving MER gives developers a shorter path from finished code to merged work. It means they can stay focused, spend more time building, and see more of their work land in customer hands.

//The rising cost of delivery, and the way through

The same validation bottleneck limiting throughput is also changing the economics of software delivery. As AI increases code volume, every inefficient loop in the delivery system gets more expensive: more CI runs, more waiting, more agent retries, and more compute spent before a change reaches production.

This chapter starts with the CI cost of getting one change to main. Then it widens the lens to the larger cost picture, including why total CI spend can still rise even when teams become more efficient, how AI token usage magnifies the cost of slow feedback, and why moving routine validation into the inner loop is becoming the practical path to controlling delivery costs.

09 Why MER matters: The 2026 cost imperative

MER does more than move throughput. It drives the thing engineering leaders are most worried about this year: what it costs to ship software when code volume keeps climbing.

Every extra validation cycle costs money. When a change runs CI on a feature branch, that run uses compute. If it fails and runs again, the team pays for more compute. At low volume, the expense may be negligible. But at AI scale, even fractions of a penny add up fast. This is where MER moves from a simple productivity metric to a serious cost signal.

To test that assumption, we analyzed average CI credit spend on main-branch and feature-branch workflows for the median team and the Top 20 cohort.

We multiplied each group's average feature-branch workflow cost by the number of feature-branch runs reflected in its MER to estimate pre-merge CI cost. Then we added the cost of a typical main-branch run to approximate total CI cost per shipped change.

If MER affects cost, teams with lower MER should spend fewer CI credits to ship each change. Here's what we found:

- The Top 20 cut their CI compute per shipped change by 31% year over year. That tracks with the MER improvement we saw earlier: their MER fell 21%, from 1.62 to 1.28, meaning they needed fewer feature-branch runs to get each change onto main.
- The median team moved the other way. Their CI credit usage per shipped change rose 13% year over year, while MER stayed almost flat. In other words, the typical team is not getting meaningfully more efficient, and each shipped change is now costing them more in CI compute than it did a year ago.
- The Top 20 use approximately 51% fewer CI credits for every shipped change than the median team, not because they are doing less effective CI, but because their MER is lower.

As AI pushes more code into the delivery pipeline, merge efficiency has a direct effect on CI spend. Teams that reduce unnecessary feature-branch cycles can ship more work with less CI compute per change. Teams that do not will pay more for each change they manage to ship.

10 The flip side of unit economics

Improving MER clearly lowers the unit cost of software delivery. When teams move changes through the pipeline with fewer validation cycles, each shipped change consumes fewer CI credits.

But lower cost per change can also make room for much higher change volume. That creates the other side of the cost story: even when each change gets cheaper, the total bill can still rise if the team ships far more work.

Consider the Top 20 cohort

Top 20 metric	YoY Change
Main-branch throughput	+72%
CI credits per shipped change	-31%
Total CI credit spend	**+62%**

This group used their more efficient validation pipelines to increase main-branch throughput by 72% year over year. They also drove down their cost per shipped change by reducing MER. But because they shipped substantially more work, total CI credit spend still rose 62%, even with a 31% improvement in credits per shipped change.

That is the flip side of better unit economics. Efficiency does not always shrink the total bill. Increased delivery costs can be justified when they reflect more valuable work reaching customers. The warning sign is rising delivery spend without a matching increase in useful output.

Anatomy of a $900k validation bill

CI credits are the first visible cost of a failed validation cycle. In agentic workflows, failed validation also burns a much more expensive resource: AI tokens.

Consider a 50-developer team working at an agentic pace on a medium-sized codebase. The team:

- Ships roughly 3,000 changes per month (about 3 changes per developer per day)
- Matches the typical MER, with each shipped change requiring roughly 5 total CI runs (4 on feature branches and 1 on main).

The cost problem comes from the retry loop. When an agent works on a change, it loads the repo context it needs to reason about the code, say 200,000 tokens for a typical mid-sized project. That context sits in a cache the agent can reread cheaply on each turn, as long as the cache stays warm.

But the cache expires after a short idle period. If a CI run takes 5 to 10 minutes to come back with a failure, the agent has been sitting idle long enough that the cache goes cold. So when the failure finally arrives, the agent can't reread the context at the cheap cached rate. It has to reload all 200,000 tokens at the cold-input rate, roughly 10x more expensive, then inspect the logs, generate a patch, and run validation again. Every slow round trip pays that reload penalty. After 5 total loops to ship the change, the cost can reach $25 or higher:

Cost component	Modeled cost per shipped change
Initial repo context load, about 200K tokens at $6.25/M	~$1.00
Additional input tokens from 3x cold reloads, CI logs, and retry history	~$12.00
Output tokens from code generation and rework at $25/M	~$9.00
CI compute cost across 4 feature-branch cycles + 1 main workflow	~$3.00
Total	~$25.00

At 3,000 shipped changes per month, that translates to about $75,000 per month, or roughly **$900,000 per year** in delivery costs. For a project requiring 500K input tokens or more, common for larger production codebases, the same pattern crosses **$1.5 million.**

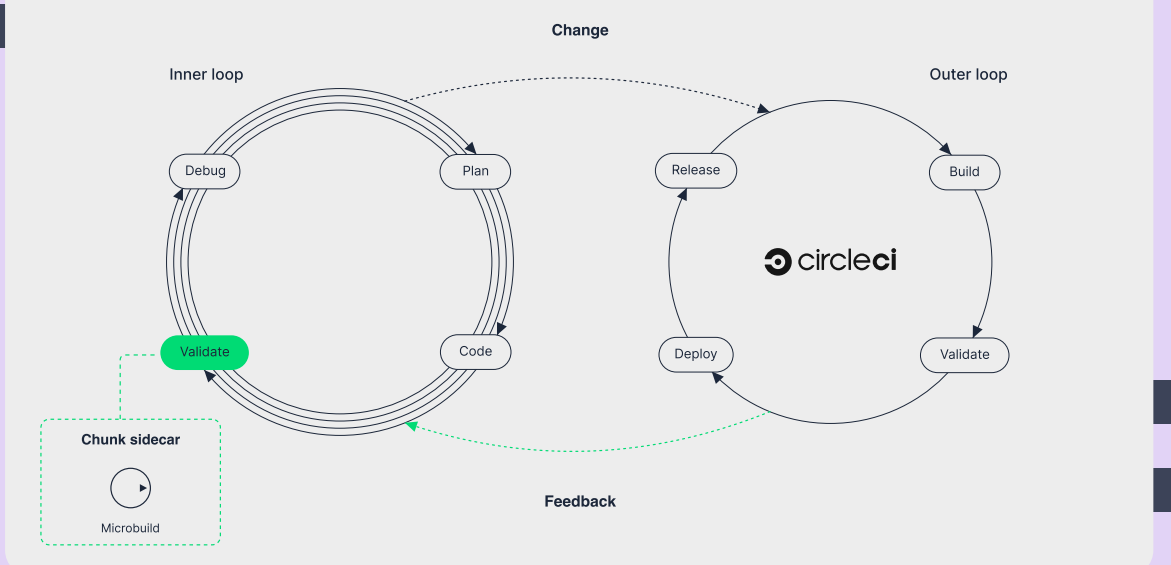
Actual costs will vary by model, context size, cache behavior, workflow duration, and retry rate. But the point of this model is to show how slow validation changes the economics of delivery. At AI-driven volume, slow feedback loops can consume a meaningful share of the delivery budget.

12 Validation in the inner loop

The $900K model points to a straightforward fix: catch routine failures in the inner development loop, before they reach CI and the AI prompt cache expires. That means moving fast checks into the developer or agent workflow before code is pushed. Instead of waiting several minutes for CI to catch a syntax error, lint failure, or broken unit test, the agent gets feedback in seconds and can fix the issue while the repo context is still loaded.

Validation layer	Feedback time	Best used for
Inner loop	<30 seconds	Lint, syntax, unit tests, small build checks
Outer loop	5-10 minutes	Integration, security, deployment readiness

This is how teams break the delivery bottleneck created by AI code volume. Routine failures get resolved before they reach CI. Fewer changes get stuck cycling on feature branches, fewer retries burn tokens and CI credits, and more AI-generated code makes it out of branches and onto main.



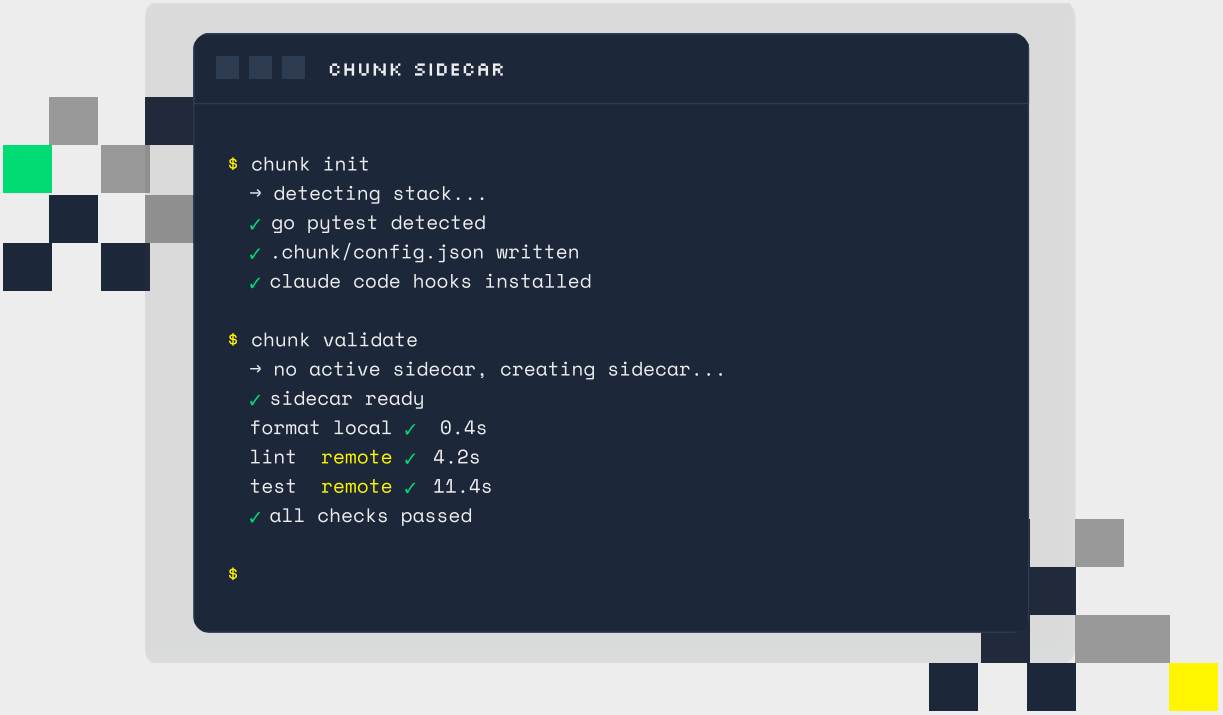
//Closing the loop

The data reveals a clear pattern: top teams lower MER to ship more validated code with fewer wasted CI cycles and lower cost.

The hard part is implementation. Most teams do not have a simple way to catch routine failures before code reaches CI. This chapter shows how teams can move validation before the push, reduce unnecessary CI runs, and break the validation bottleneck without building custom tooling.

13 Introducing Chunk sidecars

Chunk sidecars give teams a cleaner way to run validation before code reaches CI. Each sidecar is a secure Linux microVM connected to the project workspace, so an AI agent can test current changes, including unstaged edits, before committing them. It runs lightweight microbuilds against the checks teams already depend on—linting, unit tests, build checks, and configuration validation—all within the agentic loop rather than after the push.



In our testing, sidecar microbuilds changed the feedback loop in two measurable ways:

- **Compute efficiency:** Microbuilds return targeted feedback on a change in **under 30 seconds**, compared with about **5 minutes** for a full CI run. That makes routine validation up to **30x more compute-efficient.**
- **Token efficiency:** Microbuild output was about **5x more token-efficient** than standard CI logs because it returned focused failure context instead of full pipeline output.

With sidecars, teams can move routine validation into the inner loop, keep unnecessary retries out of the outer CI loop, and give more changes a cleaner path to main.

14 The bottom-line impact of inner-loop validation

Moving early validation into the inner loop via Chunk sidecars fundamentally changes the economics of agentic software delivery.

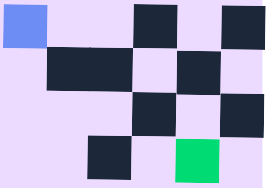
Because microbuild feedback returns in seconds, the agent's cache stays warm between iterations. This drops input token costs from $6.25 per million down to the **$0.50 warm-cache tier.**

Per change, the optimized breakdown speaks for itself:

- **AI token inputs** (warm-cache hits plus noise-free context): $1.50 (down from $13.00)
- **AI token outputs** (minimal agent rework from precise logs): $4.20 (down from $9.00)
- **Compute Cost** (27-second microbuilds vs. full pipeline runs): $0.30 (down from $3.00)
- **Total Cost per Shipped Change: $6.00** (down from $25.00)

For that same 50-developer team shipping 3,000 changes a month, the monthly bill falls from $75,000 to **$18,000,** and annual delivery spend drops from roughly $900,000 to about **$200,000.**

That's around **$700,000 a year back** in the budget, achieved purely by matching the feedback architecture to the modern pace of agentic work.



15 Conclusion

Software delivery has shifted from a code generation problem to a code validation problem. AI makes writing code faster, but legacy push-to-CI workflows are not built for machine-driven volume. The result is pipeline gridlock, wasted retries, and rising costs.

The teams pulling ahead are optimizing how AI-generated code gets validated and merged. Moving routine checks into the inner loop gives agents faster feedback, keeps unnecessary retries out of CI, and helps control delivery spend.

Ready to break the validation bottleneck?

Chunk sidecars are available now to all CircleCI users. Sign up for CircleCI to try Chunk sidecars and bring fast microbuilds into your AI development loop.

//Methodology

To create this report, we analyzed data from over 20 million CircleCI workflows within the first 28 days of March 2026 and compared them year-over-year against the identical window in 2025.

To ensure baseline benchmarks focus on repeatable software delivery, we restricted population-level metrics to projects with more than one contributor and workflows that ran at least 4 times. For metrics tracking aggregate volume, these filters were omitted to preserve data integrity and ensure full organizational activity was accurately captured. Standard data hygiene exclusions were universally applied across all metrics to remove test accounts, abusive projects, and internal CircleCI activity. Branch designations reflect the branch status at the time each pipeline ran. Industry data is sourced from Clearbit and is not available for all organizations.

+20M

CircleCI workflows within the first 28 days of March 2026

Data details:

- Every day between March 1, 2026 and March 28, 2026 (compared YoY to March 1–28, 2025)
- Over 20 million workflows evaluated
- Population metrics restricted to projects with >1 contributor and workflows with ≥4 runs
- Volume-based metrics utilize full organizational data

Contributors:

Report authors: Jacob Schmitt, Ron Powell
Report editor: Gillian Kieser
Report designers: Toni Cater, Birdy Wheeler-Wolowicz, Ruby Jazz
Acknowledgments: Bijan Samimi, Klimis Tsakiridis