// WHITE PAPER · PUBLISHED 8 AUGUST 2026

What AI First Actually Costs

A field report from one developer, four repositories, and every meter in the stack read out loud

Jason Newell · MAX Research Collective · Report WPR-BIZ-001 · Published 8 August 2026
Licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0),
https://creativecommons.org/licenses/by/4.0/
Original: https://www.maxresearchcollective.com/articles/what-ai-first-actually-costs.html
Cite as: Newell, J. (2026). What AI First Actually Costs. MAX Research Collective, WPR-BIZ-001.
https://www.maxresearchcollective.com/articles/what-ai-first-actually-costs.html

Ninety pull requests used to be a month

On a Tuesday afternoon the review tool went quiet in the middle of a pull request. Nothing was down.

Ninety reviews had gone through in the trailing seven days. The vendor's published fair usage ladder had stepped hourly capacity from ten reviews down to one, served one at a time. For weeks that event sat in the notes as an outage.

It was the plan working as designed.

All that had changed was the rate at which one person opened pull requests, and the ladder that answered for it had been sitting in the published policy the whole time.

A pull request is the unit a change travels in. It carries a branch, a diff, its tests, a description, an automated build, a review, and a merge. Ninety of them is a respectable month for a small team. Most tooling and most pricing were designed around teams like that.

Two repositories under one developer, running a disciplined agent loop, produced 317 in sixteen days. That's close to twenty a day. Ninety now arrives before the end

of the first week. Table 1 carries the counts, enumerated from the repositories rather than estimated.

Table 1. Pull request volume, enumerated from the repositories.

MEASURE	VALUE
Code pull requests per month, primary repo	about 280
All pull requests per month, primary repo	about 405
Two repository total	317 in 16 days, about 600 a month
Organisation wide	450 to 600 a month
Sustained burst, second repo	101 in 5 days
Fleet, eleven day window	243 opened, 235 merged, 193 reviewed
Developers	one

Table 1 is the complete data set and takes no appendix.

Nothing about the unit changed. These are ordinary pull requests, carrying production grade enterprise software to production.

What moved was the rate, by something between five and ten times. That one fact does most of the damage in what follows, and almost none of the damage lands where a budget review would look for it.

What had to exist before the first line shipped

Handing a developer a model license is a procurement event. On its own it produces nothing, and calling it an AI first strategy doesn't change that.

The gap between the license and the first production grade pull request is filled with work, and that work has a bill. Table 2 inventories what had to exist here before an agent loop could be trusted with enterprise software. None of it is exotic. All of it

had to be authored. Most of it was authored by the agent, which means it was paid for out of the largest line in the stack.

The standards file is the place to start. It's the document a hosted reviewer reads to learn house rules, and on one repository it runs 709 lines. On another it runs 1,945. That grew from 717 in eleven days, or 2.7 times its starting size inside a fortnight. The sections added in that window came straight out of the reviewer's most repeated findings: test quality, debounce behaviour, honest diagnostics.

That growth is the asset compounding, and it's also the meter running.

Every one of those lines was written by an agent billing tokens.

Alongside it sit a 208 line reviewer configuration, five executable structural lints, a seven job continuous integration pipeline, a test triad expected on every change, a findings ledger recording who caught what, and a failure registry naming the defect classes that recur. Continuous integration, usually shortened to CI, is the automated build and test pass that runs on every change before it can merge.

Then there's a cost nobody budgets, because it has no vendor. Three separate graders now read this repository: the hosted reviewer, a bundled second reviewer, and the coding agent itself. Each reads a different file. Pointing all three at one shared body of criteria through configuration parsed cleanly and did nothing. The guideline entry turned out to be directory scoped, so it never reached the paths its rules were written for.

No configuration makes three graders read one file. So a new defect class gets written into the standards file, the failure registry, and the second reviewer's instructions. Somebody writes it three times, by hand.

Table 2. What had to exist before an agent loop could be trusted with production code.

ARTIFACT	MEASURED SIZE	AUTHORED BY	WHAT IT BUYS
Standards file, primary repo	709 lines	the agent	a reviewer that cites house rules by number
Standards file, second repo	1,945 lines, from 717 in 11 days	the agent	the same, compounding
Reviewer configuration	208 lines	the agent	path filters, per directory guidance, knowledge base
Structural lints	five ast-grep rules	a person, by hand	a defect class that cannot recur silently
Second reviewer instructions	separate file set	the agent	grounding for the bundled lane
Challenger configuration	directory plus legacy file	the agent	parity for the trial
CI pipeline	seven jobs, about 42 billable minutes	the agent	the gate everything merges through
Test triad	unit, integration, system per change	the agent	single round reviews
Findings ledger	189 rows with attribution	a person, by hand	every finder number in this article
Failure registry	named defect classes	a person, by hand	errors that stay fixed

Table 2 is the complete data set and takes no appendix. Elapsed build time was never recorded and is not estimated.

That maintenance scales with the number of AI participants rather than with headcount. Add a fourth grader and the tax goes up, while the org chart stays where it is.

None of this happens on its own either. A tool finds a defect class. A person still has to notice it's recurring, write the rule, and write the lint. Every lint in this fleet was written by hand after somebody read three findings and saw the same shape three times.

The stack, once somebody reads the meter

An earlier version of this analysis put the monthly stack at about $412. It named CI as the largest line. Both the figure and the conclusion were wrong, in a way that performed this article's own thesis on the person writing it.

Here's how. The coding agent's subscription price had been entered as the agent's cost. That happened three days later. By then two separate review vendors had each demonstrated that a subscription price and a metered reality are different numbers.

Every line in the stack got that check except the one doing the checking. Reading the account's billing history produced the correction, and billing history is where every durable number here comes from.

Three classes of number appear in Table 3; they aren't equally reliable. Measured figures were read off a billing page or enumerated from the account's own history. List prices come from a vendor's published rate and have never touched an invoice. Corrected figures are published because the first reading was wrong. Figure 1 sets the subscription column against the metered one, layer by layer.

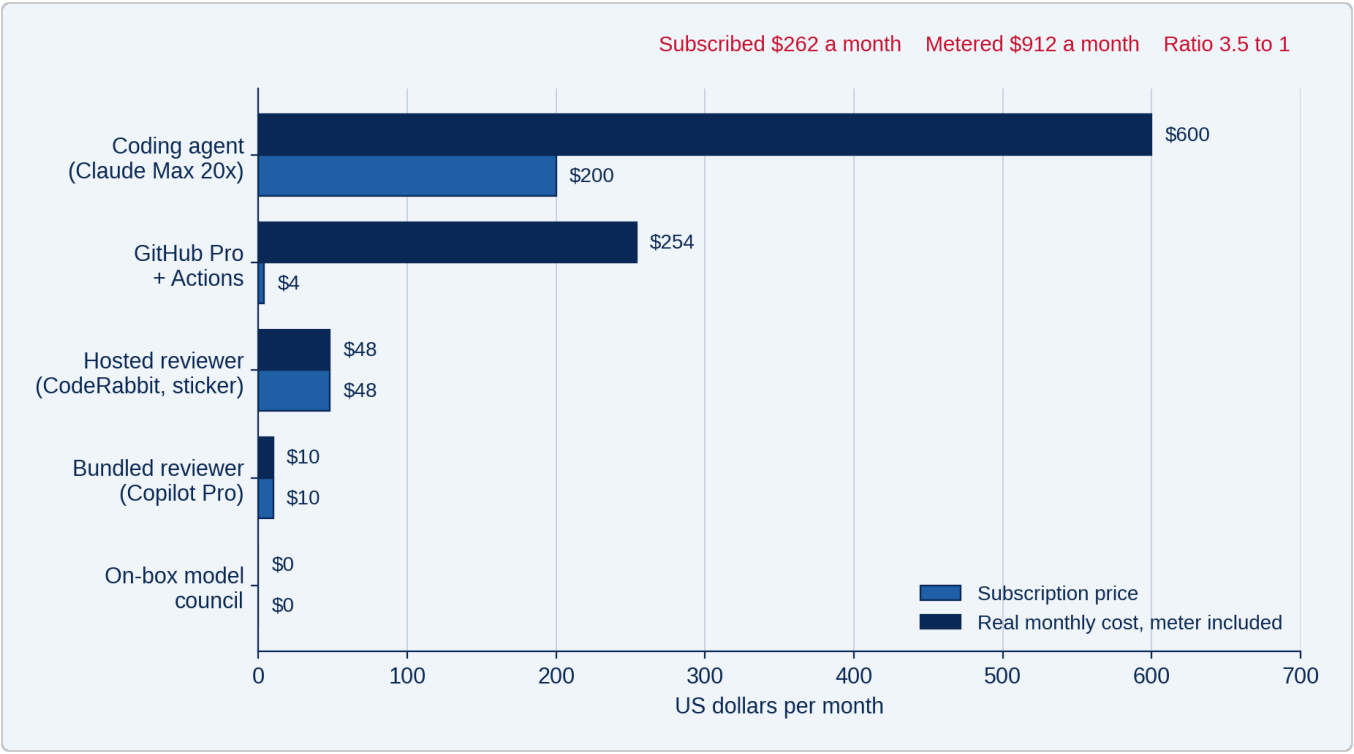


Figure 1. Subscription price against the metered reality, layer by layer. Compare the two agent bars, then the two Actions bars.

Table 3. The monthly stack, subscription price beside what the meter did.

LAYER	SUBSCRIPTION	HISTORICAL OVERAGE	REAL MONTHLY	CLASS
Claude Max, 20x tier	$200	about $400	about $600	measured, billing history
GitHub Pro plus Actions	$4	about $250	about $254	measured, billing history
CodeRabbit Pro Plus, sticker	$48	$0, capped at $1	$48	measured, plan page
GitHub Copilot Pro	$10	$0, budget set to zero	$10	measured, account meter
On box model council	$0	none	$0	measured
CodeRabbit at demanded rate	$48 sticker	queue about 1,500 deep	$660 to $2,713	calculated, see Table 6
Total, queue accepted	$262	about $650	about $912	corrected

Table 3 is the complete data set and takes no appendix. An earlier total of about $412 is superseded. Pricing and cost figures come from trials, plan pages, and billing history on the dates given, and may vary by vendor, region, seat count, and contract term.

Against $262 in subscriptions, the real stack runs about $912 a month. A budget review would have seen the smaller number. It’s off by a factor of three and a half, which is $3,144 a year against $10,944.

The largest line is the agent at roughly $600, where overage alone ran near twice the subscription. That means the included allowance was exhausted well before month end, every month. CI is second at about $254.

The hosted reviewer reads as $48, and that figure needs the same suspicion as every other sticker in this table. It buys about 720 reviews a month against roughly 2,243 demanded. So $48 is the price of accepting a queue about 1,500 reviews deep. Serving the demanded rate costs between $660 and $2,713, depending on the route taken. Latency is doing the paying, and latency never files an invoice.

So the honest ranking puts the model first, the compute that tests its output second, and review third only while the queue is tolerated. AI assisted development is expensive because the AI is expensive. That naive assumption holds at this volume, and the earlier draft was wrong to invert it.

Ranking matters less than the common driver. Every overage in Table 3 traces to the same source. The agent writes the tokens, triggers the CI runs, and generates the review rounds. One throttle upstream therefore moves every number downstream, which is the only lever in the stack that touches all of them at once.

The quality layer sitting on the pyramid

CI is the largest variable line, so it shouldn't be a hand wave. One real run, measured as wall clock per job, splits into four backend test shards at 6.9, 8.1, 9.5 and 10.6 minutes. A frontend job takes 4.0, security 3.0, and documentation detection 0.2. Billable time for the full pyramid is about 42 minutes per push.

GitHub Actions bills per job wall time, summed. Running jobs in parallel shortens the clock and leaves the bill alone. At 280 code pull requests a month and about three CI triggering pushes each, that's 35,280 minutes. Take off the 3,000 included with the plan. At eight tenths of a cent a minute, the rest lands at roughly $258.

Historical Actions overage on the account reads about $250. The account also carried a $300 overage limit at one point; nobody sets a limit of that size unless overage of that order is real. Three routes to the same neighbourhood is as much corroboration as a derived number usually gets.

Sensitivity is the useful part, and Figure 2 plots it. Two runs per pull request costs about $164. Four costs about $352. Push discipline therefore moves the CI bill by

$188 a month before any reviewer is involved. That's almost four times the reviewer's seat price, won or lost entirely on how the agent is allowed to work.

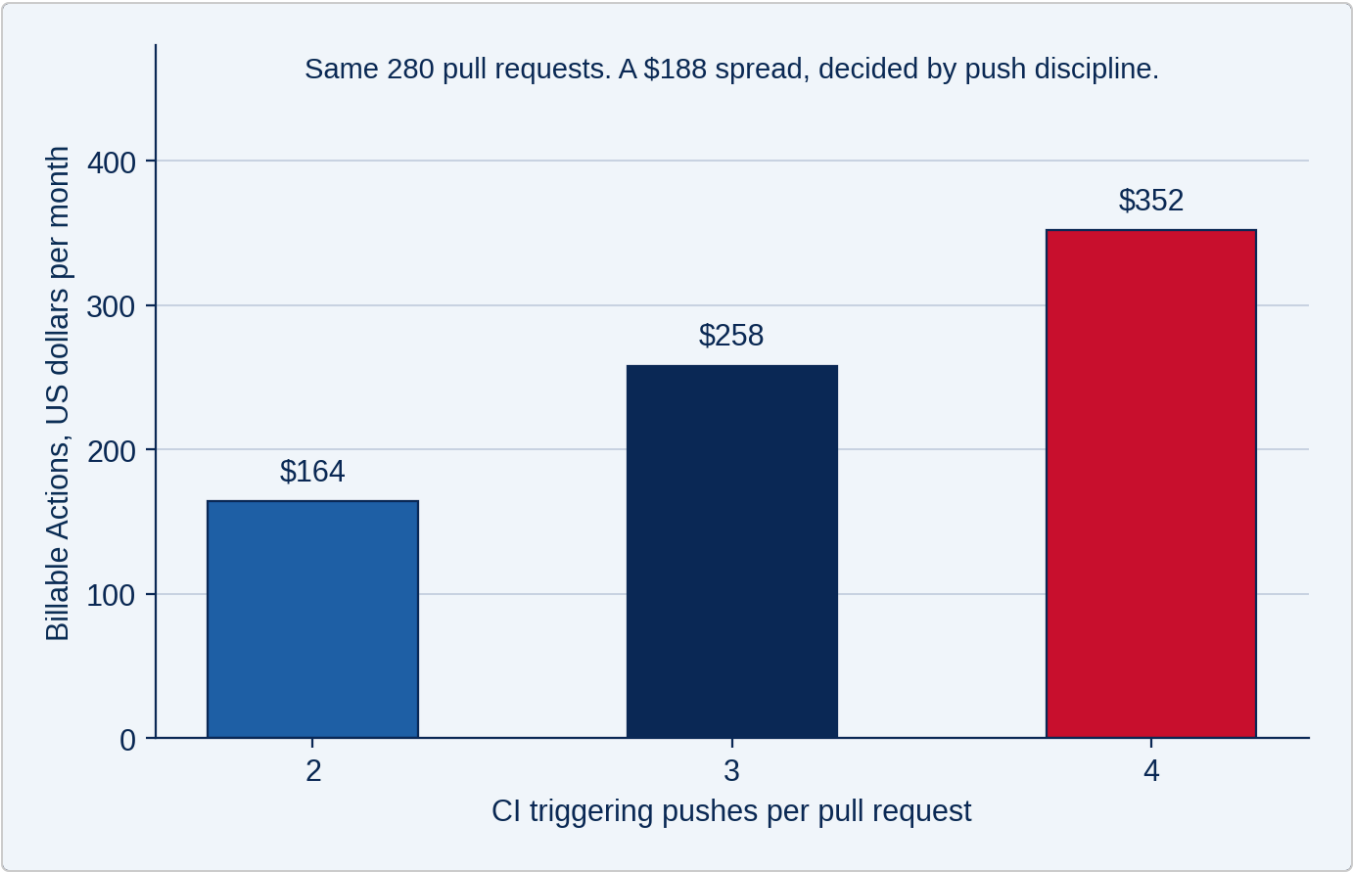


Figure 2. Billable Actions for the same 280 pull requests at two, three, and four CI triggering pushes each.

An AI reviewer sits on top of that pyramid and reads what the tests can't express. Every finding it files opens a round that costs a fix authored by the agent at token rates, a push that wakes the whole pyramid, and another forty two minutes of billable compute before anyone finds out whether the fix actually held. Across 280 pull requests a month, one extra round each is about 11,760 CI minutes, or roughly $94 of Actions on top of the tokens for every fix.

Returns run the other way, and they compound. Sixty seven real defects were caught by the incumbent reviewer over a 91 pull request tenure, at the cheapest point they will ever be caught, before merge, while the agent still holds full context. Every recurring class then gets promoted into an executable lint that runs for free

inside CI. That promotion drives the reviewer's own future round count down, so the layer gets cheaper to run the longer it runs well.

The signal ratio moved in eleven days. The first findings roll up recorded 14 actionable against 3 declined, one of which was a true false positive. The second recorded 10 and nothing declined. The third recorded 5 and nothing declined. One hard false positive in eleven days is the whole tally.

A well scaffolded reviewer converts its findings into checks. A noisy one manufactures fix cycles in the two most expensive meters in the stack.

The same $48 seat buys either outcome, and the difference between them is scaffolding somebody has to write and keep current.

What the machines found, and what the developer found

Across three weeks and four repositories, five reviewers held tenures on this fleet. Table 4 carries the record, and Figure 3 puts the finders side by side.

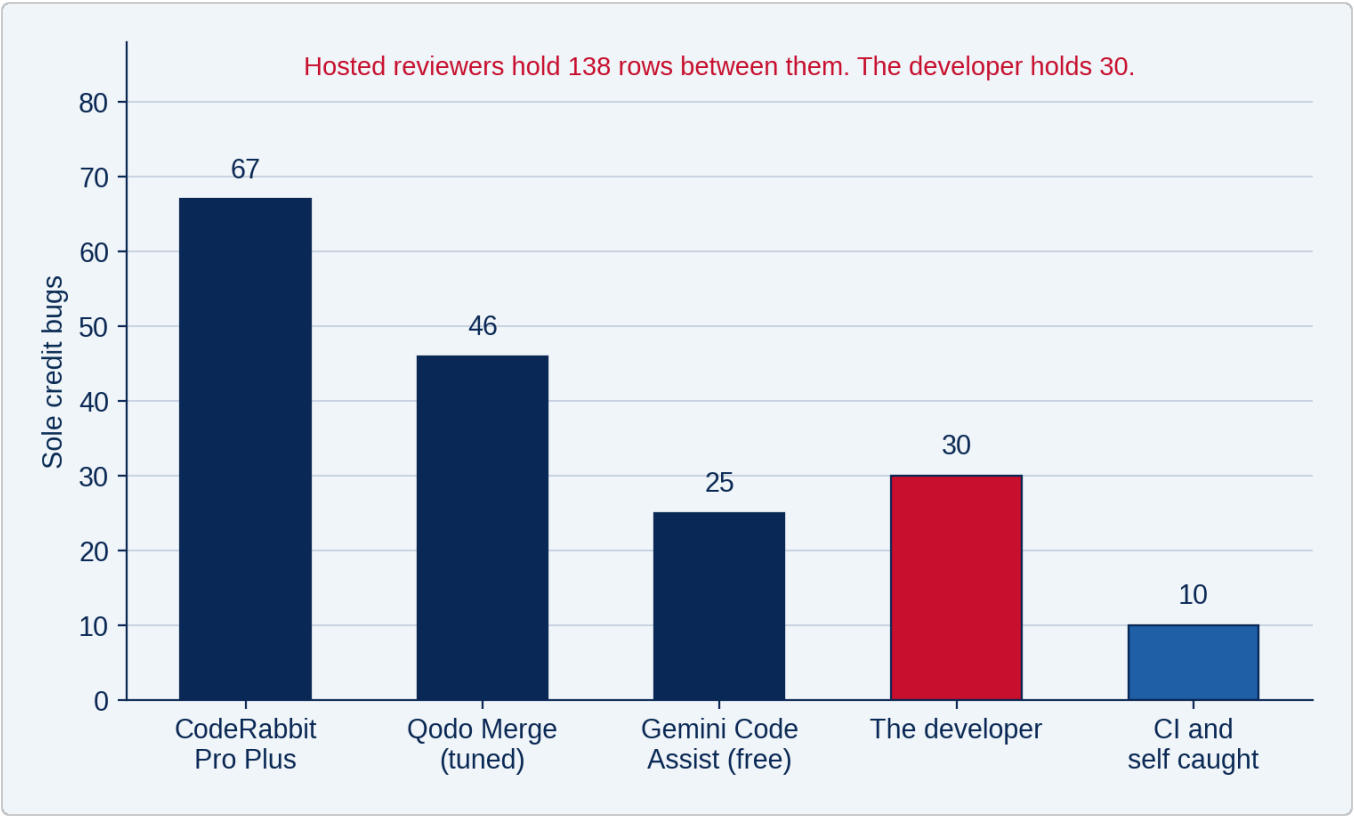


Figure 3. Sole credit bugs by finder across the July 5 to 27 arc. The red bar is the human.

Table 4. Seven finders across the 5 July to 7 August arc, and what ended each reviewer tenure.

FINDER	SOLE CREDIT BUGS	TENURE	PER 100 PRS	OUTCOME
Gemini Code Assist, free	25	about 55 PRs	45.5	free until the vendor sunset it on 17 July
Qodo Merge, tuned	46	about 44 PRs	104.5	dropped for redundancy and a pricing cliff
CodeRabbit Pro Plus	67	about 91 PRs	73.6	kept, incumbent
Greptile v5	3	2 PRs	excluded	trial closed inside two days
GitHub Copilot Pro	2, plus 2 false positives	2 PRs	excluded	kept, bundled, never chosen
The developer	30	all three phases	not applicable	job moved from finding to building the finder
CI and self caught	10	all three phases	not applicable	the free floor

Source rows and the normalisation transform are in Appendix E.

Counted by bug rather than by comment, over comparable spans of roughly a hundred pull requests each, the hosted reviewers hold 138 sole credit rows. The developer holds 30 across all three phases. The CI and self caught row holds 10.

A machine found more than twice the developer’s whole three week total in eleven days. The gap is the finding, and it’s the one worth carrying into a board conversation.

Read carefully, though, because the number cuts both ways. It doesn’t say the developer got worse. It says the developer’s job moved. Finding defects by reading diffs stopped being the highest value use of the hour, and building the apparatus that finds them took its place. Every lint, every standards rule, and every ledger row

that made those 138 findings possible came from a person reading three reports, recognising the same shape in all of them, and deciding the pattern was worth encoding permanently.

Coverage held while that happened. In the eleven day window the fleet opened 243 pull requests and merged 235. The reviewer engaged on 193 of them, about 91 percent of the 211 that touched code. Per repository it ran 86 percent, 96 percent, and 100 percent. A fourth repository sits at a deliberate zero, because the reviewer is switched off there on purpose.

Speed had a hole underneath it. First response landed between two and six minutes on the high volume repositories, with a sampled median of 2.7. The CI gate that blocks a merge sits at ten minutes, so the reviewer cleared it with room. One repository's median pull request lifetime is 13 minutes, though, and 55 of 100 merged inside 15. So the reviewer is racing the merge. It lost once, on a pull request that merged on the walkthrough before its ten findings arrived.

Two structural findings belong here more than any bug count.

The first is about running more than one lane. Two reviewers with near disjoint output, overlapping around two percent, are purely additive. On one pull request two vendors filed the same defect twenty four seconds apart, neither able to see the other's output. One reviewer flagging an error is an opinion somebody has to evaluate. Two of them, independently, inside half a minute, is close to a fact.

The second is about how easily a bake off measures the wrong thing. In the first phase, 394 findings were rated by hand across 110 pull requests. One tool posted 172 findings at 80 percent actionable. The other posted 222 at 27 percent, and the decision looked made. Then the clock got read. The second tool was arriving about thirty minutes late, after the fixes had landed. It was commenting on code that no longer existed. Tuned, its actionable rate ran near 100 percent, and it went on to post the highest normalised score in the entire record.

That evaluation nearly cancelled the better tool for arriving second.

Nobody's pricing model survives this workload

Three pricing shapes met this volume. All three broke. Table 5 sets their terms side by side.

Table 5. Six vendors, three pricing shapes, and where each one breaks at this volume.

TOOL	BASE PRICE	INCLUDED	AT THE LIMIT	TO CONTINUE	BILLABLE UNIT
Claude Max 20x	$200 a month	allowance gone before month end	agent stops	overage, ran about 2x subscription	tokens
GitHub Pro plus Actions	$4 a month	3,000 Actions minutes	CI gate disappears	$0.008 a minute	billable job minute
CodeRabbit Pro Plus	$48 a developer, $576 a year	unlimited reviews, rate limited	throttles to 1 an hour, never stops	$0.25 a file, optional	file, only if forced
Greptile Pro	$30 a seat	50 credits a seat, unlimited repos	full stop	$1 a credit, required	review
Qodo Merge	about $30 on promotion	documented standard of 20 PRs a seat a month	promotion cliff	23 to 30 seats, quoted near $2,040	pull request per seat
Gemini Code Assist, consumer	free	everything, documentation included	no limit met	shut down July 2026	none
Gemini Code Assist, enterprise	$45 to $54 a month	licensed, with limits	not trialed here	next bake-off	not established
GitHub Copilot Pro	$10 a month	1,500 AI credits	review and editor both end	$0.01 a credit, varies by model and repo size	credit

TOOL	BASE PRICE	INCLUDED	AT THE LIMIT	TO CONTINUE	BILLABLE UNIT
Greptile local CLI	free	run before a pull request opens	not established	unknown, may draw the hosted allowance	none hosted

Table 5 is the complete data set and takes no appendix. Prices are list prices except CodeRabbit and Copilot, which are read off this account. The Greptile overage rate is low confidence: the meter was read rather than the price of the fifty first review. Pricing and cost figures come from trials, plan pages, and billing history on the dates given, and may vary by vendor, region, seat count, and contract term.

Seat pricing assumes seats track work. At agent velocity one seat generates what a room of developers used to. One vendor’s promotional unlimited tier sits on top of a documented standard of twenty pull requests per seat per month. Honouring that standard at 450 to 600 organisation pull requests a month means 23 to 30 seats for one person. The quote came back near $2,040 a month. That’s roughly seventy times the incumbent line for identical work.

Per unit pricing is unbounded against a workload that manufactures units. The metered challenger was modelled at two to three reviews per pull request, which is what a human round looks like and what every engineer’s intuition agrees with. The meter counts pushes. An agent pushes a fix, then the test for the fix, then a documentation correction, then a lint cleanup. Metered across two pull requests, eighteen pushed heads produced sixteen credits. Call it 0.89 per push. The estimate was five times low, and no amount of rereading the pricing page would have caught it.

One pull request consumed 8 of 50 included credits, sixteen percent of a month’s allowance on a single change. A fourteen day trial therefore expires in about a day and a half of ordinary work, which means the trial ends by exhaustion rather than by calendar.

Flat pricing protects itself with a throttle, so unlimited becomes a latency budget. Figure 4 plots the published ladder. At roughly 65 code pull requests a week, the trailing seven day count crosses ninety sometime on Tuesday. Then it stays there.

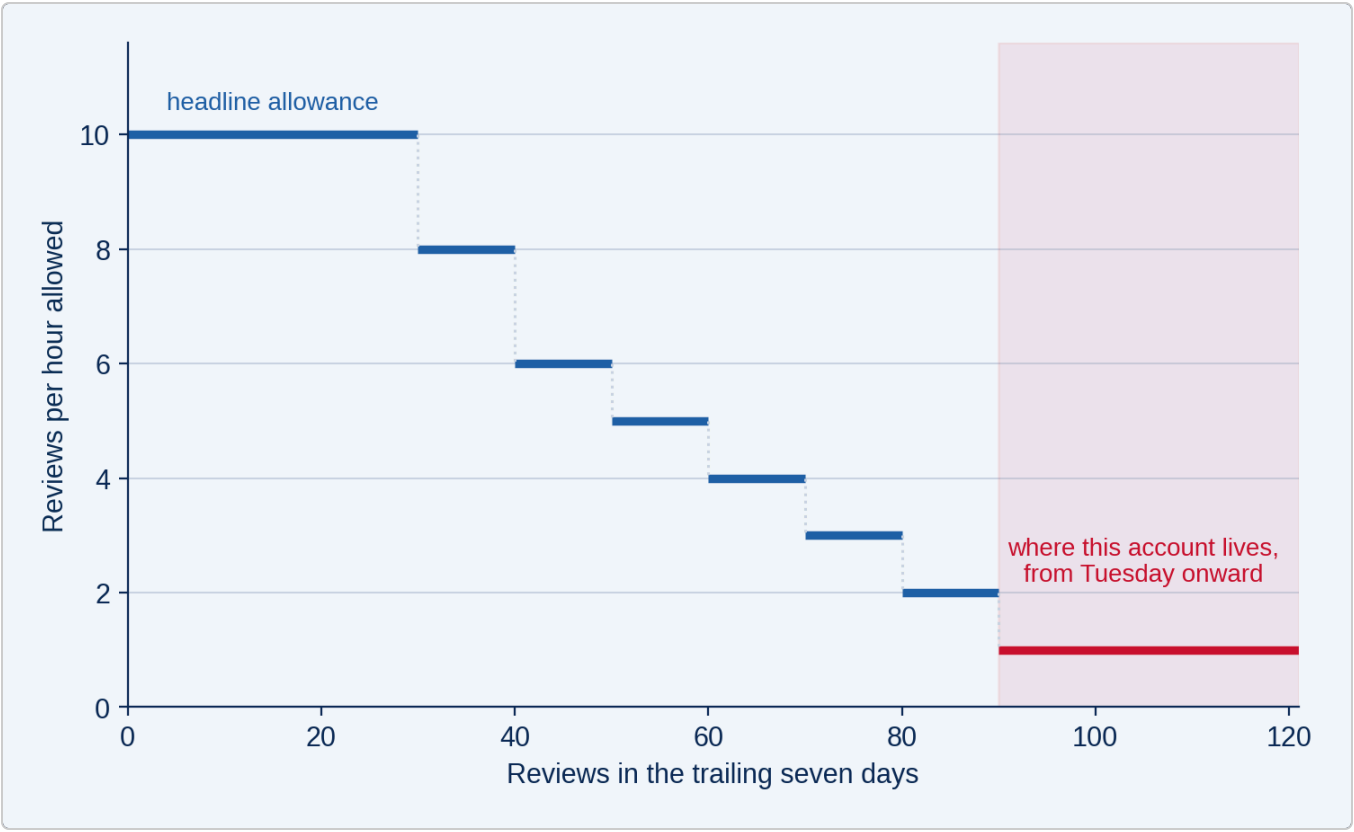


Figure 4. The published fair usage ladder. The shaded band is where this account sits from Tuesday onward.

That lands at one review an hour, served one at a time, or about 168 a week against roughly 260 demanded. Throttling is a slowdown, and the distance from a stop is where the money sits. Even on the bottom rung the seat keeps delivering, continuously, at roughly 720 reviews a month for no marginal cost.

The waiting is real and it was measured. Rate limiting hit about six times in one roll up. It then produced 13 minute waits across roughly 15 reviews on 9 pull requests, and hit twice on a single pull request after that. No finding was ever lost to it. That makes it a measurement of a commit habit as much as a vendor cap, since pushing five small fixes into an open review means paying for five reviews and reading one.

Then there's the tell that a pricing model has lost contact with what it sells. Included credits scale per seat, while overage costs more per credit than the seat does. Fifty credits bundled with a $30 seat prices a credit at sixty cents, against a dollar for overage. So the cheapest way to buy 997 credits is twenty seats nobody needs, at $600 against $977, a saving of thirty nine percent.

It looks absurd on an invoice, and procurement will ask about it, and it remains straightforwardly cheaper. Whenever a plan's included allowance scales per seat and the overage rate exceeds the per seat unit rate, buy seats until the allowance covers usage.

None of this gets gentler with scale. Fifty developers working this way make the throttle ladder useless and the per unit invoice unbounded at the same time, and the seat model's absurdity multiplies rather than amortising.

Why the challenger trial ended in two days

A fourteen day trial isn't fourteen days at this velocity. Greptile includes fifty reviews with a seat. Two pull requests consumed sixteen of them, read off the vendor's own billing page.

That made it a six pull request trial, in a repository merging about fourteen code pull requests a day. The clock that ran out was the meter, and any trial advertised in days tells you nothing about sample size when the vendor's unit is one an agent loop generates continuously.

Three structural facts ended it. None of them are about analysis.

Pricing is the first, and the section above covers it. The model prices the unit this workflow produces most of, and the estimate drawn from the published page ran five times low.

The second is a default that declines rather than partially reviews. Out of the box the tool refuses any pull request above four changed files, and the message it returns names the count: eighteen files against a four file limit. It's configurable and it got raised. The default is the finding.

An agent that fixes a bug, adds its test, updates the documentation and refreshes a derived artifact has touched four files before anything unusual has happened. For a comparison that's worse than it is for a workflow, because a low finding count can mean a decline rather than a pass, and those are two different facts with only one of them about review quality.

The third is a skip rule that couldn't skip. Both reviewers were configured to ignore documentation only pull requests. The incumbent's skip is a keyword gate the platform evaluates before dispatching the model, so it skipped a documentation pull request and posted a status saying so within seconds. The challenger's skip lives in prose instructions the model reads after dispatch.

Reading the rule is the run. It reviewed the same documentation only pull request four times.

A cost control that costs isn't a cost control, and under per review billing that distinction is most of the product.

Delivery belongs here too, stated precisely. On the one pull request where both saw the same diff, the inline comment ratio was 45 to 1. That measures delivery mechanism rather than analytical quality. The challenger's analysis was frequently sound, and it covered documentation files the incumbent's filters exclude, which is real coverage this shop is otherwise missing.

It arrived as summary prose, some of it written into the pull request body rather than the review surface, where nothing can be replied to, resolved, or tracked as an item. For an agent working the loop unattended, a finding that can't be addressed programmatically has to be read by a person first.

What the trial was measured against got fixed before any data arrived, so it couldn't be fitted afterwards. The challenger replaces the incumbent only on more sole credit real bugs at the same commit hashes, and on a cost path not an order of magnitude worse at this volume. It cleared neither. Three bugs across two pull requests is far too short a tenure to normalise, and the cost path is an order of magnitude worse by construction rather than by accident.

Three limits sit on that verdict and belong beside it. Two pull requests, one repository. Judged out of the box against an incumbent tuned for weeks with executable structural lints, an asymmetry that flatters this account rather than the challenger. And the overage rate is low confidence, because what got read was the meter rather than the price of the fifty first review.

Then the outcome that survives all of it. The same vendor ships a free local command line tool. Run before a pull request opens, it costs nothing hosted and drops into exactly the position a shift left pass already fills. Whether local runs draw against the same allowance is unconfirmed, and that one question decides whether the cheapest useful thing in this entire evaluation is free.

That was the fourth reviewer tenure on these repositories. One ended because the vendor withdrew the product. One ended on portfolio redundancy while holding the highest normalised find rate in the record. One ended on pricing structure and two defaults. Only one of the four was decided on findings.

At agent velocity, what selects your review layer is increasingly something other than how well it reviews.

What one review costs, and what it's worth

Every vendor sells reviews. No two mean the same thing by the word. The gap between those meanings is where the money goes. Table 6 normalises it. It carries what each tool counts as one unit, how many units a single pull request here consumes, and what that works out to per code pull request and per month.

Table 6. What one review means at each vendor, and what a pull request here actually consumes.

ROUTE	WHAT ONE UNIT IS	UNITS PER PR	PER CODE PR	PER MONTH	CLASS
CodeRabbit, queue accepted	one review, up to 300 files	9 pushed, about 3 served	$0.17	$48	measured plus policy
CodeRabbit, forced through	$0.25 a file past the rung	9, at about 7 files	$9.69	$2,713	calculated, never billed
Greptile, seats for credits	1 credit, measured 0.89 a push	about 8	$4.82	$1,350	list price
Greptile, seat plus overage	same, $1 past the first 50	about 8	$7.94	$2,223	list price
Qodo, enforced standard	1 PR a seat a month, capped at 20	1, then a new seat	about $7.29	$2,040	vendor quote
Copilot Pro	1 AI credit	not measured	unknown	$10 to day three	measured allowance
Gemini Code Assist, consumer	no unit	unlimited	$0	$0	withdrawn July 2026
Greptile local CLI	no hosted unit	run pre pull request	$0 hosted	$0 hosted	free tool, allowance draw unconfirmed

Table 6 is the complete data set and takes no appendix. Per month figures assume 280 code pull requests. Pricing and cost figures come from trials, plan pages, and

billing history on the dates given, and may vary by vendor, region, seat count, and contract term.

Read the queued row against the metered row. Same repository, same diffs, same developer. A factor of twenty eight sits between them, and nothing about the work changed. One vendor counts pushes and the other counts patience.

That's the whole reason a sticker price misleads. A number that sounds identical across two quotes describes two different quantities, and no amount of comparing the quotes surfaces it. Metering one real pull request does, in about an hour.

Price is one axis. Table 7 puts it against what each reviewer actually caught, using sole credit bugs per hundred pull requests scaled to the 280 a month this shop opens.

Table 7. Cost set against findings, using sole credit bugs scaled to 280 pull requests a month.

ROUTE	BUGS PER 100 PRS	BUGS A MONTH	COST A MONTH	COST PER REAL BUG
Gemini Code Assist, consumer	45.5	127	$0	$0, product withdrawn
Qodo Merge, promotional	104.5	293	about $30	$0.10
CodeRabbit, queue accepted	73.6	206	$48	$0.23
Qodo Merge, enforced standard	104.5	293	$2,040	$6.96
CodeRabbit, forced through	73.6	206	$2,713	$13.17
Greptile v5	sample of 2 PRs	not computable	$1,350 to $2,223	not computable

Source rows and the scaling transform are in Appendix I. Pricing and cost figures come from trials, plan pages, and billing history on the dates given, and may vary by vendor, region, seat count, and contract term.

The grid is uncomfortable. The best tool on quality is the wrong buy at this volume, because its pricing model detonates at seventy times the incumbent line. The cheapest per bug was the free one, and it no longer exists in that form. The tool that survives is third on findings and first on total cost, and it wins on a throttle policy rather than on analysis.

One caution belongs on the quality axis. Sole credit counting penalises a reviewer working beside another and flatters one working alone. The tuned challenger earned its 46 while a second reviewer was also running, so its rate is understated and its quality lead is wider than the table shows. That strengthens the finding rather than softening it, since the tool with the strongest case on merit is the one nobody can afford.

Every one of these numbers multiplies

One developer is the whole sample here. That’s the weakness of it and also the use, because nothing in this stack is shared. The agent bills per person, the reviewer bills per seat, and CI minutes get consumed by whoever pushed the commit.

So every figure in this article is a per engineer unit price. Table 11 carries it against headcount, and Figure 6 plots the same thing.

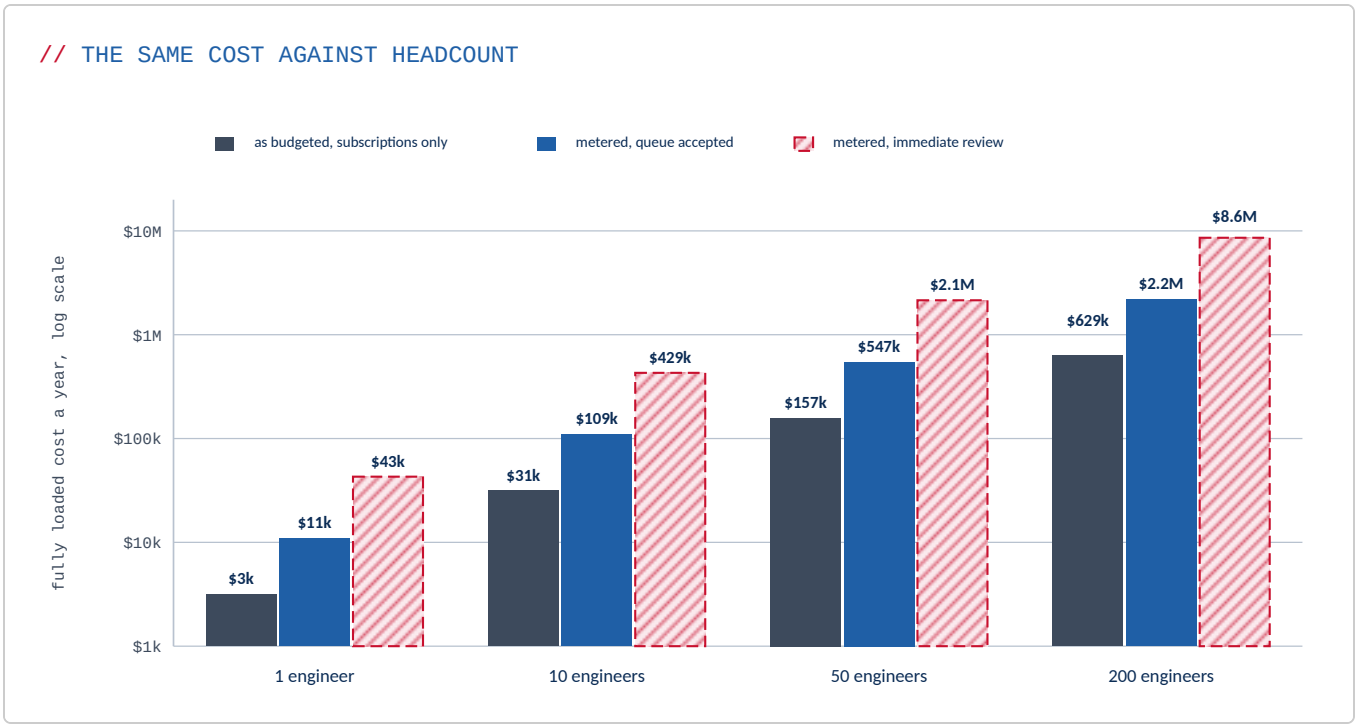


Figure 6. The same cost against headcount, on a log scale. Every line in the stack is per engineer.

Table 11. The same stack multiplied by headcount, annual.

ENGINEERS	AS BUDGETED	METERED, QUEUE ACCEPTED	GAP	METERED, IMMEDIATE REVIEW
1	$3,144	$10,944	$7,800	$42,924
10	$31,440	$109,440	$78,000	$429,240
50	$157,200	$547,200	$390,000	$2,146,200
200	$628,800	$2,188,800	$1,560,000	$8,584,800

Table 11 is the complete data set and takes no appendix. Scaffolding is per repository and does not multiply. Whether included CI minutes scale per seat was not measured. Pricing and cost figures come from trials, plan pages, and billing history on the dates given, and may vary by vendor, region, seat count, and contract term.

At ten engineers the gap between the budgeted number and the metered one is $78,000 a year. At fifty it's $390,000 a year. At two hundred it passes one and a half million, and none of it appears in the procurement decision that started the whole thing. The impatient configuration at two hundred engineers runs past eight and a half million a year, for the same code.

Two things bend that line, one in each direction.

Scaffolding is the cost that doesn't multiply. A standards file, a lint set, and a CI pipeline are per repository rather than per person. Fifty engineers write them once and amortise them across everybody. It's the line nobody budgets and the only one that gets cheaper per head as the team grows.

Included allowances bend the other way, and this is a real unknown rather than a rhetorical one. The three thousand CI minutes here are one account's allowance. Whether included minutes scale per seat on a team plan was never measured, and if they don't, the CI line grows worse than linearly, because every engineer past the first bills from their first minute.

The throttle is the one thing that improves with size. It's published per developer per hour, so a fifty person team buys fifty ladders rather than sharing one queue. Flat pricing scales linearly on price and holds its latency, which is the opposite of how most people expect a rate limit to behave.

What this pricing is and isn't

Every price here came from a trial, a plan page, or a billing history on one account, on the dates given. Vendors change tiers, run promotions, withdraw products, and quote differently by region, seat count, and contract term, which is why a rate card copied out of somebody else's field report is worth less than an afternoon spent metering your own repository against your own volume. Two of the prices in this article moved while it was being written, and one product disappeared entirely.

Treat the figures as a worked method rather than a rate card. What travels is the unit question, the queue question, and the multiplication by headcount. The dollar amounts don't. That last part includes the numbers on this article's own cover.

Three axes, and nothing wins all three

Price alone recommends whichever tool finds least. Quality alone recommends whichever tool has the worst pricing model. Both of them together still miss the constraint that decides whether a reviewer matters at all, which is whether it arrives before the merge.

So there are three axes. What it costs, what it catches, and how fast and how often it delivers. Table 8 carries velocity on its own, because two different things hide inside that word.

Table 8. The velocity axis: how fast a reviewer answers, and how much of the demand it will serve.

TOOL	MEDIAN FIRST RESPONSE	SUSTAINED THROUGHPUT AT BASE PRICE	SHARE OF DEMAND	BEATS A 13 MINUTE MERGE
Gemini Code Assist, consumer	2.2 minutes	no limit ever met	100 percent	yes, withdrawn July 2026
CodeRabbit Pro Plus	2.7 minutes, range 2 to 6	about 720 reviews a month, 1 an hour on the bottom rung	32 percent	yes at first response, not once queued
Qodo Merge, tuned	about 5.2 minutes	20 pull requests a seat a month	7 percent of 280 PRs	yes
Greptile v5	not measured	50 credits a seat, about 56 reviews, then a full stop	2.5 percent	not established
GitHub Copilot Pro	posted 24 seconds after the incumbent, one observation	1,500 credits, cleared on day three	about 10 percent of the month	yes, while credits last

Table 8 is the complete data set and takes no appendix. Demand is about 2,243 reviews a month. On an asset heavy repository the incumbent ranged from 7 minutes to 1 hour 46, so latency scales with diff size. Pricing and cost figures come from trials, plan pages, and billing history on the dates given, and may vary by vendor, region, seat count, and contract term.

First response is the one everybody quotes. On the high volume repositories the median ran 2.7 minutes for the incumbent and 2.2 for the free tool that no longer exists. The merge gate sits at ten minutes. On an asset heavy repository the same tool ran from seven minutes to an hour and 46, so latency scales with diff size rather than holding flat.

Sustained throughput is the one nobody quotes. It’s where a flat plan and a metered plan diverge completely. At its base price the incumbent serves about 720 reviews a month against roughly 2,243 demanded. Call it 32 percent. One seat of the metered challenger serves about 56, or two and a half percent. The enforced standard on the tuned tool covers twenty pull requests a seat a month, which is seven percent of the 280 this shop opens.

Those are the same fact stated three ways. Every vendor’s base price buys a fraction of one developer’s demand, and that fraction runs from two and a half percent up to thirty two.

A collision hides inside that, and the measured window doesn’t resolve it. One repository’s median pull request lifetime is 13 minutes, with 55 of 100 merging inside 15. A first response arriving in three to six minutes beats that comfortably. A review waiting its turn on the bottom rung of the ladder, behind a queue about 1,500 deep, does not. Coverage held at 96 percent through the window and no finding was ever lost to the throttle, so the collision stayed theoretical. It won’t stay theoretical at a higher volume or a longer queue, and nothing here measures where it breaks.

Table 9 puts all three axes in one grid. Figure 7 plots them together, with cost across the bottom, findings up the side, and the share of demand served as the size of each marker.

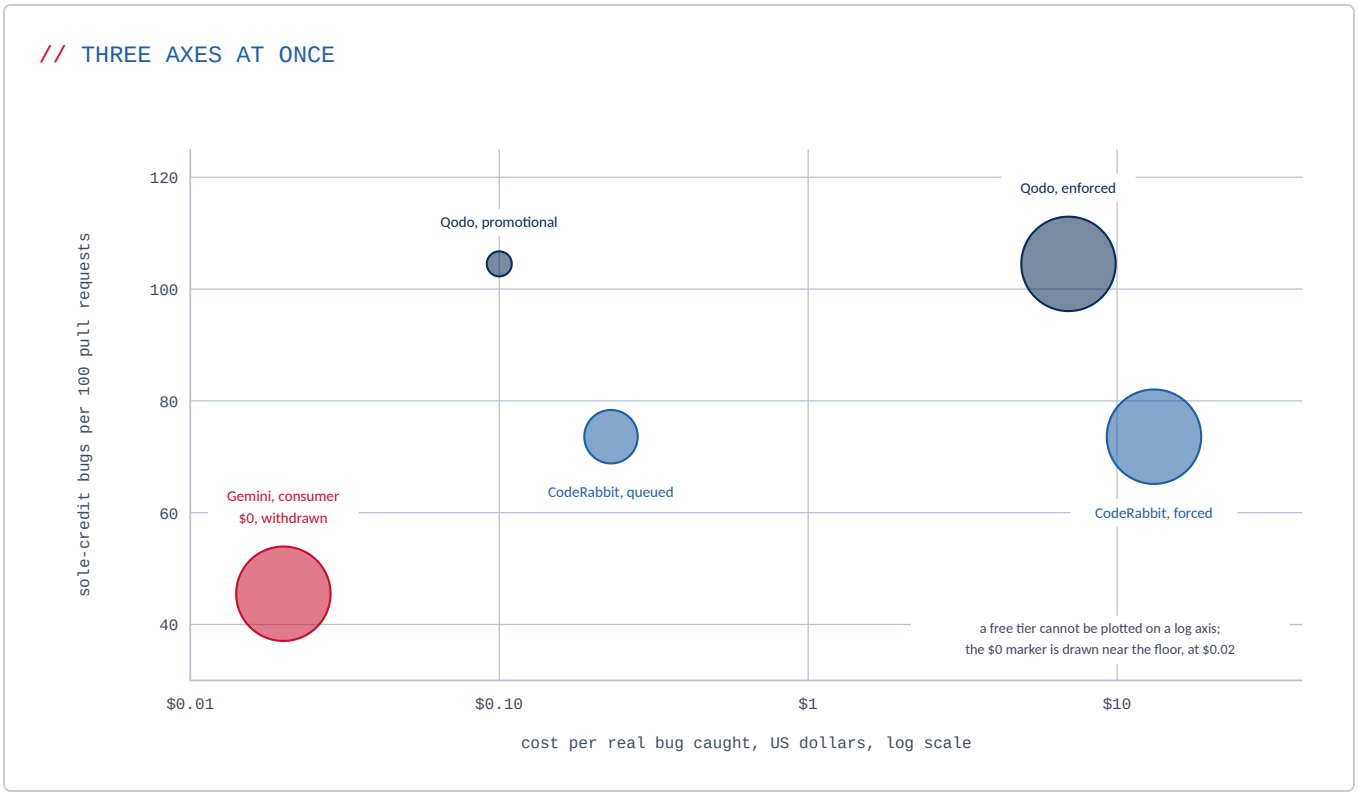


Figure 7. Three axes at once. Cost across the bottom, findings up the side, and marker size for the share of demand served.

Table 9. All three axes in one grid. No row wins all of them.

ROUTE	COST PER REAL BUG	BUGS PER 100 PRS	MEDIAN FIRST RESPONSE	SHARE OF DEMAND SERVED
Gemini, consumer	$0	45.5	2.2 minutes	100 percent, product withdrawn
Qodo, promotional	$0.10	104.5	5.2 minutes	7 percent
CodeRabbit, queued	$0.23	73.6	2.7 minutes	32 percent
Qodo, enforced standard	$6.96	104.5	5.2 minutes	100 percent at $2,040 a month
CodeRabbit, forced through	$13.17	73.6	2.7 minutes	100 percent at $2,713 a month
Greptile v5	not computable	sample of 2 PRs	not measured	2.5 percent
Copilot Pro, bundled	not computable	sample of 2 PRs	sub minute, one observation	about 10 percent

Source rows for every column are in Appendix I. Pricing and cost figures come from trials, plan pages, and billing history on the dates given, and may vary by vendor, region, seat count, and contract term.

Nothing wins all three. The cheapest per bug caught was also free, fastest of the field, and unlimited, and its consumer tier was withdrawn in July 2026 by a vendor who owed this account nothing and gave six days of notice to a shop that had built its documentation coverage around it. The strongest on findings, by a margin of about forty percent, serves seven percent of demand at its cheap price and costs seventy times the incumbent line at its enforced one. The tool that survived is first on latency, second on throughput, third on findings, and the cheapest thing left that still exists.

That's the shape of this market. Optimise any single axis and you pick a tool that fails on another, and the failure surfaces months later as a queue, a cliff, or an invoice nobody modelled.

Two of the axes move against each other under load, and that part generalises past these vendors. Paying more buys throughput and leaves findings flat, since the same reviewer does the same reading either way. Paying less buys the same findings later, until later is after the merge, at which point the findings are worth nothing at all.

What a fully loaded license costs

One developer, four repositories, eight hours a day, production grade enterprise software. Table 10 carries the fully loaded cost of that license under each decision available, and Figure 5 plots the same thing against the rest of the stack.

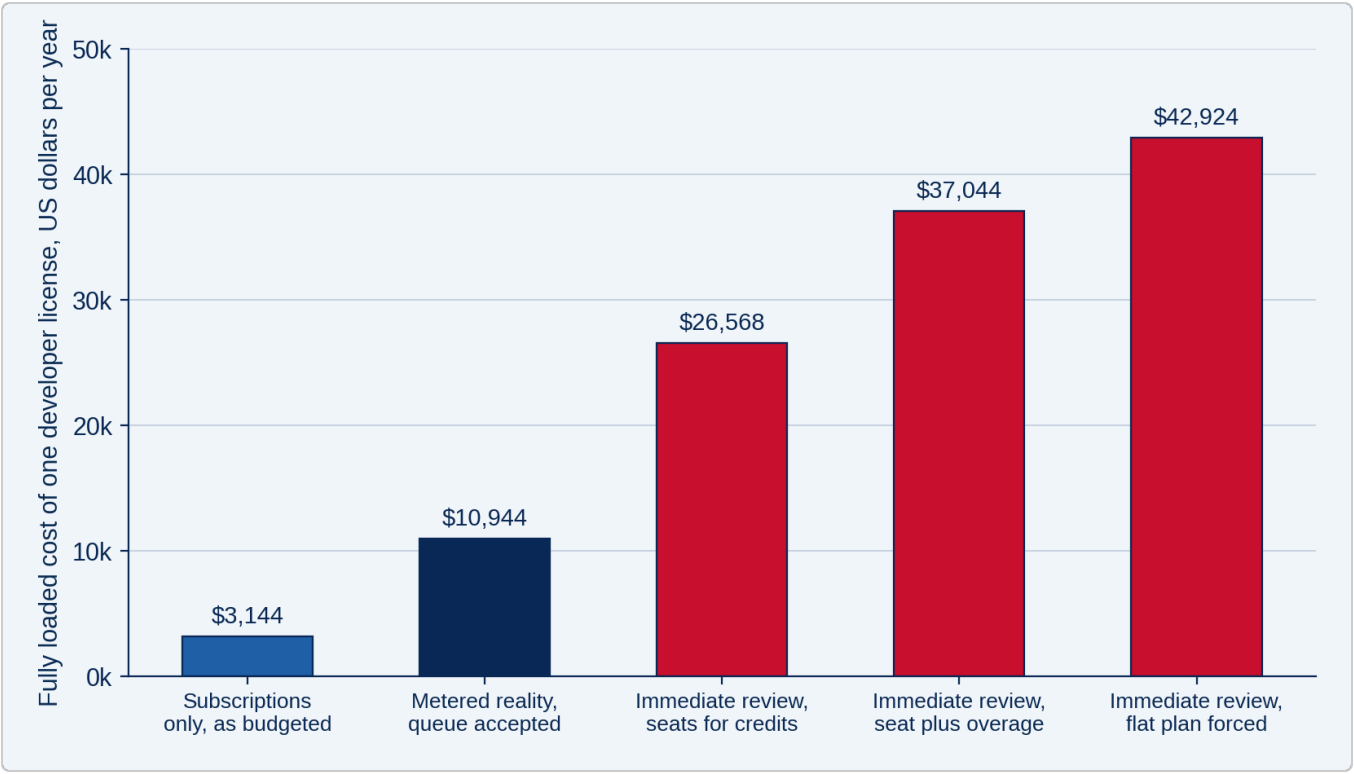


Figure 5. Fully loaded annual cost of one developer license under each decision available.

Table 10. Fully loaded cost of one developer license, at both measured review rates.

DECISION	REVIEWS A MONTH	REVIEW LAYER	STACK A MONTH	STACK A YEAR
Subscriptions only, as a budget review sees it	not applicable	$48 sticker	$262	$3,144
Metered reality, queue accepted	720 served free	$48	$912	$10,944
Immediate review, seats for credits, low band	about 1,120	$660	$1,524	$18,288
Immediate review, seats for credits, high band	about 2,243	$1,350	$2,214	$26,568
Immediate review, seat plus overage, high band	about 2,243	$2,223	$3,087	$37,044
Immediate review, flat plan forced, high band	about 2,243	$2,713	$3,577	$42,924

Source rows and every step of the arithmetic are in Appendix F. Pricing and cost figures come from trials, plan pages, and billing history on the dates given, and may vary by vendor, region, seat count, and contract term.

Patience is the variable here. Accepting the queue holds the whole stack at about $912 a month, which is $10,944 a year, and the reviewer’s sticker price is five percent of it. Refusing to wait, by any route, makes the review layer rival or exceed the $600 agent that writes the code, and takes the annual number to somewhere between $26,568 and $42,924.

The person is the same, the output is the same, and the 280 pull requests are the same. Pricing structure meeting volume accounts for the whole spread.

One caution on the review counts underneath those figures, because it’s the sort of thing that should be stated rather than smoothed. Two independent measurements of review triggering pushes per pull request disagree by a factor of two, at roughly

four and roughly nine. Table 10 carries both bands rather than picking the one that argues better. The lower band is the friendlier case and the conclusion survives it.

Normalised per reviewed pull request, the field spreads even wider. A self hosted reviewer running on an owned key works out to four to seven cents a review. The flat rate incumbent lands at nine to eleven. Per review priced tools land between fifty cents and a dollar fifty, which at this volume is $130 to $525 a month for the same work the flat seat does for $48.

The cheapest option in the field isn't hosted at all, and it costs CI minutes and maintenance instead of a subscription. It stays in the repositories as break glass, because a reviewer running on an owned key can't be sunset by a vendor.

That last point earned its place the hard way. The first automated reviewer here was free, competent, and the only one that reviewed documentation only pull requests. Google shut the consumer tier down in July 2026, which is a dependency with no contract behind it, and the replacement search happened on somebody else's schedule. An enterprise tier still runs under licensing at $45 to $54 a month with its own limits. It hasn't been trialed here, and it's the first candidate in the next bake-off.

Capping every meter buys a calendar problem

Every metered line in this stack now fails closed. Agent overage is disabled. Actions overage sits at zero, the bundled reviewer's pay as you go budget is zero, and the hosted reviewer is capped at a dollar.

That takes $912 down to $262, a cut of seventy one percent. It also converts a cost exposure into a capability exposure on every line.

CI is the one that bites. Three thousand included minutes at the measured burn rate lasts days rather than weeks. After that, pull requests merge on reviewer opinion alone. On the day this correction was written, CI caught a real defect that both hosted reviewers had passed over.

A bill that can't surprise you is worth real money, and a fail closed fleet is a coherent posture. It has to be adopted with the consequence said out loud. At agent velocity, capping a metered input truncates the month rather than the work. The layers truncate in sequence with CI first, so the last pull requests of any month are the least gated ones.

The lane nobody chose

A third reviewer runs on this repository and nobody chose it. It arrived with a plan upgrade at $10 a month, carrying 1,500 AI credits and a pay as you go budget set to zero. Past the allowance it bills a cent a credit, and how many credits a task costs depends on the input, the output, the model, whether the work runs as an agentic workflow, and the size of the repository.

The standing portfolio rule here is at most two review lanes, at most one of them paid. If this lane's review draws on those metered credits, then the count is two paid lanes, the cap is broken, and no decision was ever made. It came bundled.

The meter makes it urgent. 894 of the 1,500 credits went in two days. That's sixty percent of a month's allowance inside a fifteenth of the month. At that burn the allowance is gone on the third day, and twenty seven days of the month then have no bundled reviewer in them.

With the overage budget at zero, exposure never reaches an invoice. Exhausting the allowance instead ends code review and editor assistance together, silently, for whatever remains of the month, which is a capability cliff wearing the costume of a cost control.

A second trap sits beside it and costs nothing until it costs everything. The hosted seat is licensed to one login. A pull request opened under any other account goes silently unreviewed, with no warning anywhere in the interface. Every coverage number in this article depends on that not having happened.

The shape every one of these errors had

Going back through the mistakes this evaluation recorded turned up one defect class repeated six times. Every instance was a state meaning "this could not be observed" reported as "a bad outcome was observed."

Two interfaces returned empty, and the reading was that a reviewer had stayed silent. It had posted thirteen times. The surface it chose was the pull request body, which no workflow watches. A rate limit throttle was read as the tool stopping, when it keeps serving about 720 reviews a month at a slower cadence. An Actions query returned a 403 and the poll reported still running, when the token lacked the scope and the poll could never finish. A workflow query scoped to one file returned an empty list, which got read as no such runs exist. It was structurally incapable of seeing a differently named workflow, and an entire cost mechanism sat unexamined for a day.

In all six a tri-state of succeeded, failed, and cannot be determined got collapsed into a boolean, and in all six the third state was sitting there in the response where anybody who went looking could have read it.

Why it belongs in an article about pricing: this is the same failure as pricing a subscription while the meter sits unread. Both times the easy observation stood in for the load bearing one. Both times the substitution was invisible, because the easy observation returned a clean, confident looking answer. A 403 looks like a no, and a subscription price looks like a cost.

An observation that couldn't be made is not a negative observation, and any automated check worth having reports three states rather than two.

Questions worth asking before the AI first decision

Table 12 carries the ones that cost real money here, ordered by what they would have saved. Item one can't be answered by asking a vendor, and it pays for itself immediately. Meter a single real unit of work, end to end, with the agent working the way it actually works, before extrapolating anything.

Table 12. Questions that cost real money here, ordered by what they would have saved.

ASK THIS	WHY IT COSTS MONEY	WHAT A USABLE ANSWER SOUNDS LIKE
Meter one real unit of work before extrapolating.	A page derived estimate here was five times low.	One pull request, end to end, with the agent working the way it actually works.
What has to be built before the license produces anything?	Ten artifacts had to exist here first, most of them authored at token rates.	A named inventory with an owner, not a start date.
What is the billable event, and how many does the agent generate?	Per review and per file per review read alike in a sentence and differ tenfold on an invoice.	A named event, plus your own count of how many a week.
On a flat plan, what is the throughput limit and how does it degrade?	Flat pricing at volume is protected by a rate limit, and the ladder may be volume sensitive.	The rung you will live on, rather than the headline allowance.
What controls exist past the wall, and is any of them free?	A spend limit bounds damage without offering a zero cost option.	A queue you may wait out, alongside a paid fast lane.
Does the included allowance scale per seat?	Where it does and overage costs more per unit, seats are cheaper than usage.	A per credit price for both routes, compared.
Where does the reviewer's compute run?	A hosted tool can be free on its own invoice and still execute on CI minutes you own.	Vendor compute, or a workflow run in your repository.
Does the trial end by calendar or by allowance?	Fifty credits is about six pull requests at agent velocity.	A credit count you can divide by your own weekly volume.
How many graders read your code, and do they	No configuration makes three graders read one file, so every rule	A count of graders and the maintenance that

ASK THIS	WHY IT COSTS MONEY	WHAT A USABLE ANSWER SOUNDS LIKE
share one rule set?	gets written by hand three times.	comes with each.
Who holds the seat, and what happens to work opened under another login?	A pull request opened by any other account goes silently unreviewed, with no warning.	A named account, and an alert when coverage drops.
Is your skip rule a gate, or a suggestion the model reads?	One tool skipped a documentation pull request before dispatch. The other read its skip rule after dispatch and reviewed the same one four times.	Evaluated before the run starts, or evaluated inside it.
Does the configuration take effect, or does it only parse?	A schema check passed here while a body of review criteria sat inert.	Change something the rule should catch and confirm the tool catches it.
Does every automated check distinguish bad from cannot tell?	Six errors here were an unobservable state read as a negative one.	Three states reported, with the third one loud.

Table 12 is the complete data set and takes no appendix.

What this doesn't establish

Anything quoting this piece inherits its limits, so they belong in the open.

Reviewers held consecutive tenures of different lengths, on repositories that were themselves changing. This is a comparison across tenures rather than across identical diffs. Sole credit accounting penalises a reviewer running alongside another and flatters one running alone. One reviewer's sample is two pull requests, enough to establish direction and not enough for magnitude, so its per hundred figure is excluded rather than extrapolated. Two candidates were judged out of the

box against one deeply tuned, which is unflattering to the evaluation rather than to them.

The ledger writes one row per pull request, so a pull request carrying ten separate gaps still counts once. Every finder number is a floor. They're all floors in the same way, so the ratios survive even though the absolute counts understate.

The throughput ladder and the per file overage are published policy and a relayed vendor statement respectively. Being rate limited mid pull request is consistent with the ladder. No review has ever been forced through, so the forced overage figures are calculations rather than bills. Whether the bundled lane's runs draw down included CI minutes is unknown and worth knowing, since at about 45 seconds a run across 280 pull requests a month it comes to 400 minutes or more, against an allowance the CI gate already exhausts mid month. An earlier claim that hosted reviewers touch no CI minutes has been retracted rather than replaced.

Elapsed time is missing from the scaffolding section on purpose. Artifact sizes and authorship are measured, and the number of weeks between the first license and the first production grade agent pull request was never recorded, so it isn't estimated here.

Two hundred and eighty pull requests a month is unusual. The pricing unit finding generalises and the multiples don't, since a team running at twenty pull requests a month sits comfortably inside every one of these tools' included allowances and will never meet a wall of any kind.

One more limit belongs here, and it's the worst of them. An earlier version put the scaffolding at 320 lines against a measured 709, and I can't reconstruct where 320 came from. It entered by recollection, the one sourcing method this article spends a whole section warning about, and I applied that warning to vendors' prices and not to my own file. Direction matters too. Scaffolding turned out to be twice what was credited, which makes any out of the box comparison against a reviewer this heavily grounded even less fair to the challenger than the setup section already admits.

An AI coding agent doesn't change the price of anything. It changes how many of the priced thing you buy, which is why these surprises land on the finance office

rather than on the engineer who caused them. Nobody approved a price increase, and the invoice moved anyway. Under a cap the invoice stops moving and the calendar starts. CI runs out first, somewhere in the second week, and everything merged after that has been read by a reviewer working alone, without the test pyramid underneath it, on the same authority and the same green check as everything merged in the first week.

Appendix G, the record behind this article

Every figure here is drawn from an evaluation record that predates the article and was written for a decision rather than for publication.

RECORD	WHAT IT HOLDS
Reviewer participant register	all seven participants in one schema: observed, measured, billable unit, confidence, dated verdict, and limits
Phase one bake-off	394 findings hand rated across 110 pull requests, plus the review ordering confound
Three reviewer close-out	the keep or drop decision on seven metrics across four repositories
Challenger trial protocol	the controls fixed and the decision rule set before any data arrived: more sole credit bugs at the same commit hashes, and a cost path not an order of magnitude worse
Pricing unit economics	the cost thesis this article is the public face of
Findings ledger	the categorisation of what reviewers actually find, and which preventions got promoted upstream
Velocity measurements	the pull request throughput figures in Table 1
Bundled lane evaluation	four surfaces scored against the standing constraints, and the two paid lane cap question
Known failure patterns	the defect class behind the tri-state section

Two published articles cover parts of this record in more depth: one on the three reviewer comparison across four repositories, and one on the review layer at agent velocity.

Glossary

Alphabetical, plain speech. Each entry says what the thing is and why it matters here.

TERM	DEFINITION
Agent loop	A coding model working in repeated cycles of read, change, test, and push. It's what produces the pull request volume every number here rests on.
ast-grep	A tool that matches patterns in code structure rather than in text. It's how a recurring defect becomes a rule that runs for free on every change.
Continuous integration, CI	The automated build and test pass that runs on every change before it can merge. Second largest line in the stack, and the first to run out under a cap.
Credit	The billable unit on a metered tool. What counts as one credit decides the whole invoice.
Draft pull request	A pull request opened as work in progress. Reviewers skip drafts by default, which silently removes coverage in a shop that opens everything as a draft.
Fail closed	Behaviour where a component that cannot confirm a safe condition reports failure rather than success.
Fair usage ladder	A published table that lowers a flat plan's hourly allowance as recent volume rises. It's how a flat price protects itself at agent velocity.
GitHub Actions	The hosted service that runs CI jobs, billed per job by wall clock minute. Parallel jobs shorten the clock and leave the bill alone.
Head	One pushed state of a branch. Metered reviewers count heads rather than human review rounds, which is where the five times estimating error came from.
Inline comment	A review finding anchored to a file and a line, carrying a thread and a resolution state. It's the format an agent can act on without re-reading prose.
Overage	What a meter charges once an included allowance runs out. Every overage in this stack traces back to the agent's velocity.

TERM	DEFINITION
Pull request, PR	The unit a change travels in: a branch, a diff, its tests, a description, an automated build, a review, and a merge.
Rate limit	A cap on how many operations run in a window. On a flat plan it's where the vendor collects the bill, in latency rather than dollars.
Scaffolding	The standards files, path rules, lints, ledgers, and pipelines that ground an agent loop in one repository. It costs tokens to write and keep current, and it separates signal from noise.
Seat	A per person licence. Where included allowance scales per seat, seats can be cheaper than usage.
Sole credit bug	A real defect only one finder caught. It flatters a reviewer running alone and penalises one running beside another.
Test triad	The unit, integration, and system tests a change carries before it opens. Shipping all three in the first commit is what keeps a review to a single round.
Token	The unit a language model bills by. Largest line in this stack and the one driving all the others.

About this series

These evaluations run on live repositories at production volume, and every one gets published with its measurements, its confidence levels, and the things it failed to establish. The interesting findings have consistently been the ones arguing against the conclusion already reached.

CAREER & LEADERSHIP WPR-BIZ-001

// ABOUT THE AUTHOR

Jason Newell leads organizations and people in technology, and is a researcher, deep technologist and builder — from a CRAY to agentic AI. He founded MAX Research

Collective to build AI for regulated industries that runs where its owner has control, with healthcare as its strongest focus. About Jason → Connect on LinkedIn →