Techstrong
a Futurum company

DevOps Experience
2026 VIRTUAL EVENT

TECHSTRONG SPECIAL REPORT

The AI Agent Race
At the Top of the Stretch

Models, Agents, Harnesses and the Enterprise Race to Pick the Right AI System for Software Delivery

BY ALAN SHIMEL

Techstrong
a Futurum company

TECHSTRONG SPECIAL REPORT

The AI Agent Race: At the Top of the Stretch

Models, Agents, Harnesses and the Enterprise Race to Pick the Right AI System for Software Delivery

By Alan Shimel

With extensive Futurum Research contribution of data and analysis

DevOps Experience
2026 VIRTUAL EVENT

In support of DevOps Experience 2026
September 24, 2026 · devopsexperience.com

ABOUT THIS REPORT

A field guide to models, agents, harnesses and the enterprise controls that determine whether AI can deliver useful software.

PRODUCTION

Visible author: Alan Shimel
Design and production: Perplexity Computer
Published August 2026

CONTENTS

Follow the field

40.17%
AI in code generation

6.20%
AI in deployment decisions

5.84%
Autonomous end-to-end

75.21%
Reported ≥1 incident

READ THE DENOMINATOR
Traffic, survey adoption, enterprise use and benchmarks answer different questions.

EVALUATE THE SYSTEM
Model, harness, tools, context and controls determine the accepted outcome.

EXPAND IN STAGES
Increase authority only after reliability is proven inside the prior boundary.

FIGURE INDEX

The evidence at a glance

READ THE DENOMINATOR. Model traffic, developer adoption, enterprise model use, benchmark results and customer outcomes are different measures. This report keeps them separate.

THE EVIDENCE MIX

- Futurum primary research
- Observed platform activity
- Developer survey evidence
- Vendor-published customer outcomes

EXECUTIVE SUMMARY

Executive Summary

Which AI coding agent is winning?

It sounds like a straightforward question. It is not. Claude Code has raced to the front of several developer-adoption measures. OpenAI Codex is closing quickly. GitHub Copilot brings enormous distribution and a privileged position inside the repository and pull-request workflow. Cursor is trying to turn the development environment into a mission-control center for agents. Google has formidable models, cloud infrastructure and developer reach. Devin and a growing field of specialists are pursuing longer-running autonomous work. OpenHands, Cline, Kilo Code, OpenCode and Aider are making the case that the harness should remain open and the model replaceable.

The difficulty is not a lack of data. It is that the available data measures different things. OpenRouter measures traffic routed through OpenRouter. Hugging Face can identify agents that interact with the Hugging Face Hub. Developer surveys measure what their respondents say they use. Futurum surveys measure foundation models in production, organizational maturity, software-lifecycle adoption and governance. Benchmarks measure a particular model operating inside a particular harness with a particular set of tools and resource limits. Customer stories show what one organization says it accomplished with one implementation.

None of these denominators represents total coding-agent market share.

> **The market keeps trying to turn this into a model race. Software delivery makes it a systems race.**

That is the first conclusion of this report and the foundation for everything that follows. Model share is not agent share. Token share is not developer share. Benchmark leadership is not enterprise adoption. A customer success story is not a controlled comparison.

The market keeps trying to turn this into a model race. Software delivery makes it a systems race.

The model supplies intelligence, but the harness supplies context, tools, memory, permissions, execution, tests and the feedback loop. The surface determines how a developer delegates and reviews work. The control plane determines which models and tools are permitted, what the agent can touch, what it costs and who remains accountable. The performance of the whole system—not the name printed on the model card—determines whether an agent can deliver useful software.

The evidence points to a field taking shape but not yet settled. Claude Code appears to be at or near the front in observable coding-agent adoption. Codex is a leading challenger with rapid adoption growth and top-tier benchmark performance. GitHub may own the most valuable piece of track because it sits where code, workflow, security and enterprise policy converge. Cursor has become a serious agent-native development environment, supported by both adoption and notable enterprise deployments. Open harnesses are no longer fringe experiments; they offer model portability, local execution and sovereignty at the cost of greater integration and operational responsibility.

The enterprise data also shows why nobody should declare the race over. AI is used by 40.17% of respondents in code generation and 37.66% in code review, but only 13.23% in CI/CD operations and 6.20% in deployment decisions. Individual developer assistance remains the dominant mode at 47.20%. Truly autonomous end-to-end agents stand at 5.84%. Enterprises are increasingly willing to let AI write a change. They are far less willing to let it decide that the change should go live.

That restraint is understandable. In a Futurum survey of 839 software-lifecycle decision-makers, 75.21% self-reported at least one production incident to which they believed AI had contributed. Another 17.52% suspected an incident but could not confirm it. Yet only 43.27% said human code review was mandatory, 38.50% required AI security scanning and 32.06% had human approval gates for agent actions. These are self-reported findings and the underlying question did not provide enough metadata to establish independent causation. Even with that qualification, the gap between reported incidents and deployed controls is too large to ignore.

The winner of this race will not simply write the best code. It will combine task performance, enterprise context, workflow integration, acceptable economics and a level of autonomy that fits inside the organization's acceptable blast radius. That means the right choice will differ by task. The best assistant for an individual developer may not be the best background agent for a

repository, the best modernization agent for a large legacy codebase or the best agent to approach a production environment.

There may be a leader, but there will not be one horse for every course.

SECTION 1

Meet the Field

Before reading the tote board, we need to know what is actually in the race.

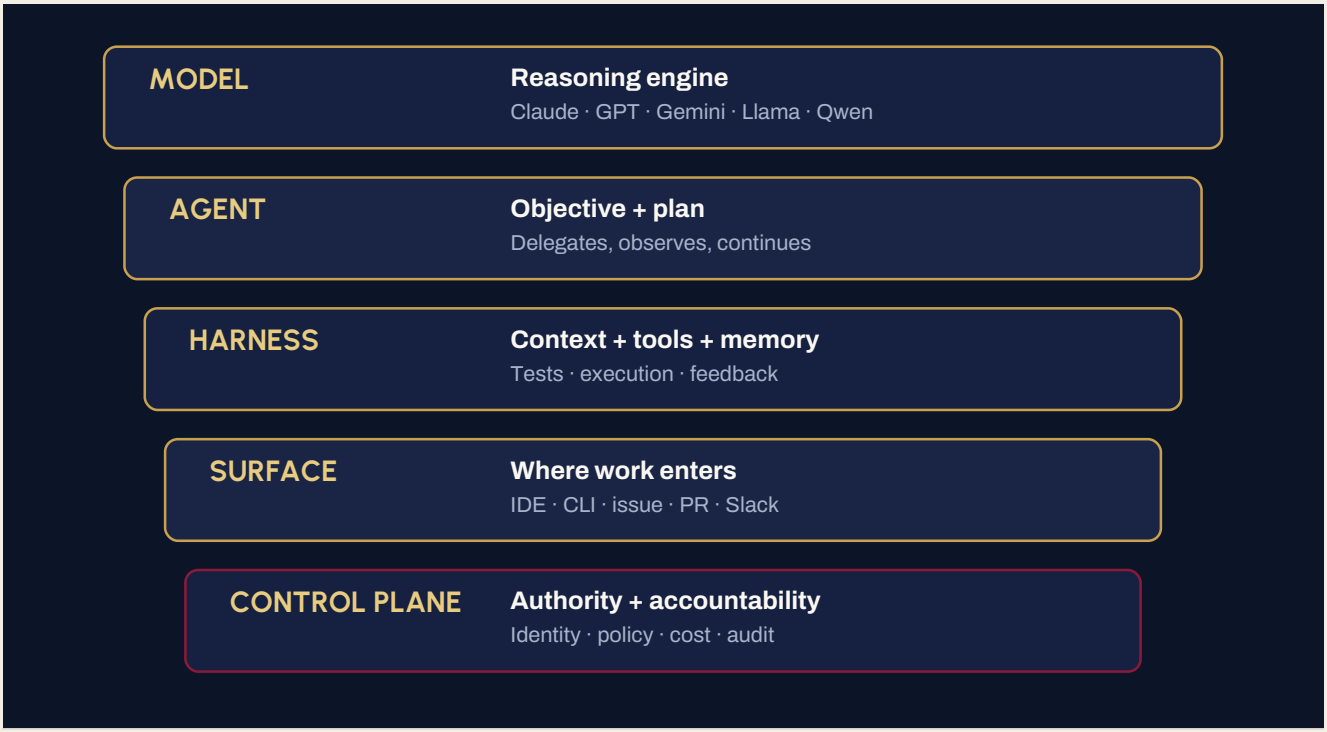
A foundation model is the underlying intelligence engine: an OpenAI model, Anthropic's Claude, Google's Gemini, Meta's Llama, Qwen, DeepSeek, Mistral or another model family. It interprets the request, reasons about the problem and produces language, code or tool calls.

An agent is the working software entity that receives an objective, builds or follows a plan, gathers context, invokes tools, observes results and continues until it completes the task, reaches a boundary or asks for human intervention.

A harness is the machinery surrounding the model. It determines how repository context is collected, which instructions are loaded, what memory persists, which tools are available, what the model is allowed to execute, how tests are run and how failures are fed back into the next attempt. Claude Code, Codex, OpenHands, Cline, OpenCode and Aider can all be understood partly as harnesses, even when they are also marketed as agents or products.

The surface is where a person encounters the agent. It may be an IDE, command line, web interface, GitHub issue, pull request, Slack conversation, ticket or background cloud environment. Surface matters because it determines how naturally the agent enters the existing workflow and how easily its work can be inspected.

FIGURE 1 **The system is the unit of competition**



Five layers separate intelligence from enterprise authority. Examples are illustrative; layer boundaries increasingly blur. Source: Techstrong analysis.

The enterprise control plane governs the entire system. It decides which models are approved, which repositories an agent may access, what data can leave the organization, which MCP servers and tools are trusted, what credentials can be issued, where human approval is required and how activity and cost are recorded.

These layers are increasingly being bundled together, which is why market comparisons become slippery. Claude is a model family, but Claude Code is an agent and harness. Codex refers to an OpenAI coding-agent product and its associated models and surfaces. GitHub Copilot began as an assistant and now encompasses multiple agentic experiences and multiple models. Cursor is simultaneously an editor, agent surface, harness and model-routing layer. An enterprise can use Claude through Claude Code, GitHub Copilot, Cursor, an API or a cloud platform. It can use OpenAI models directly or through products and platforms controlled by other vendors.

The same model can also perform differently inside different harnesses. The system prompt changes. The context selection changes. The available tools change. The time and compute budget change. One harness may know when to search, run a test and revise a patch; another may waste context reconstructing an API call or stop after the first plausible answer.

This makes the harness more than packaging. It is part of the product's intelligence utility.

SECTION 2

Reading the Tote Board

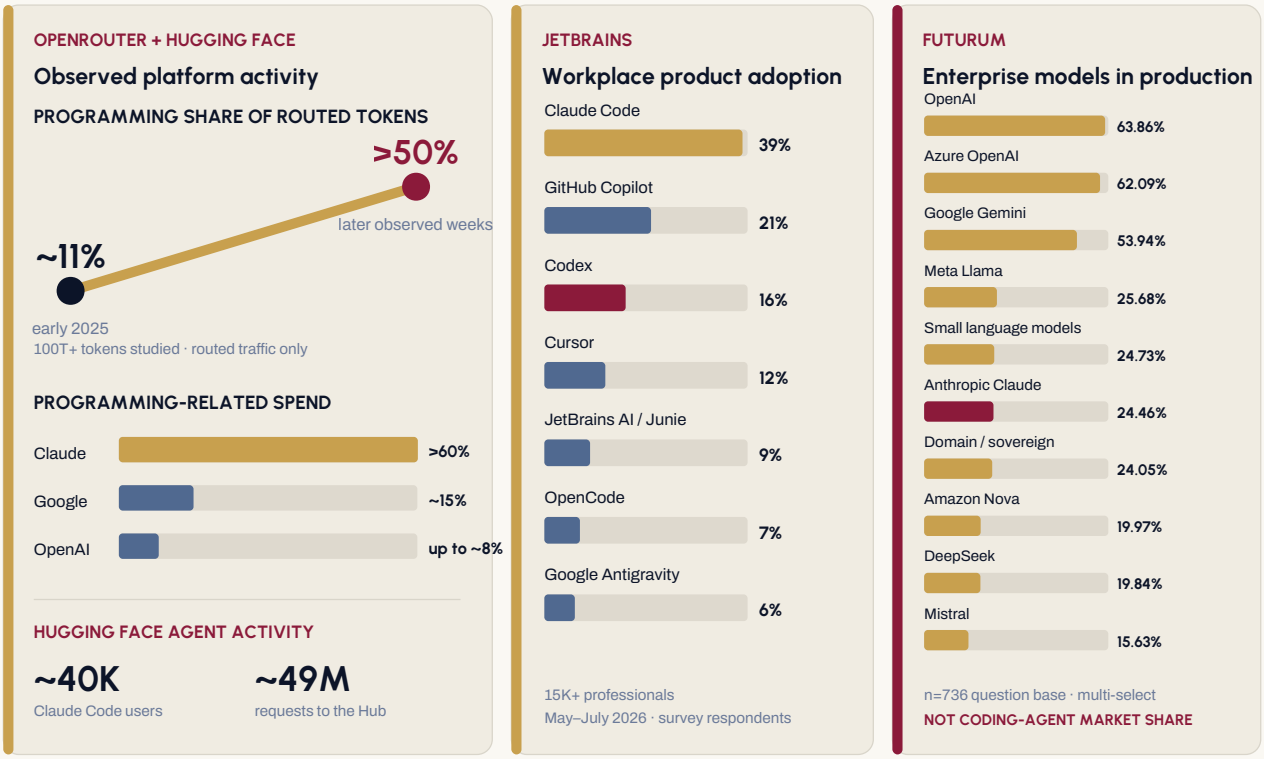
OpenRouter provides one of the largest public windows into real model usage. Its analysis of more than 100 trillion tokens found programming growing from roughly 11% of token volume in early 2025 to more than 50% in later weeks. Claude accounted for more than 60% of programming-related spend for most of the observed period, Google held around 15%, and OpenAI rose from roughly 2% to about 8% over the measured interval. Qwen devoted an estimated 40% to 60% of its own traffic to programming workloads. [P1]

Those numbers are substantial, but they describe OpenRouter. Direct subscriptions to Claude Code, Codex and other services do not necessarily traverse it. Large enterprises using Bedrock, Vertex AI, Azure or direct model contracts may also be absent. OpenRouter tells us which models developers and applications choose within a large routing marketplace. It does not tell us total enterprise share.

Hugging Face sees another slice. Since it began identifying agent traffic in April 2026, Claude Code generated traffic from approximately 40,000 distinct users and nearly 49 million requests to the Hub, with Codex close behind. Hugging Face identifies them as the two largest agents in its data, well ahead of the rest. These numbers show real agents performing real work against a developer platform, but they measure interaction with Hugging Face—not the full universe of agent usage. [P2]

JetBrains offers the broadest developer-survey view in this report. Its 2026 Developer Ecosystem Survey covered more than 15,000 professional developers globally. Between May and July, 90% said they used AI coding agents at work at least weekly, and 68% said they used them daily. Claude Code was used at work by 39%, up from 18% in January. Codex rose from 3% to 16%. GitHub Copilot stood at 21%, Cursor at 12%, JetBrains AI and Junie at 9%, OpenCode at 7% and Google Antigravity at 6%. JetBrains reported Claude Code as the most-used coding tool for 31% of developers, while Codex awareness rose from 27% to 65%. [P3]

FIGURE 2 **Four windows into the market**



Three denominators produce three different rankings. OpenRouter and Hugging Face show platform activity; JetBrains shows reported workplace product adoption; Futurum shows multi-select enterprise model use. They must not be placed on a shared market-share axis. Sources: OpenRouter; Hugging Face; JetBrains; Futurum [F1].

On this denominator, Claude Code is the adoption leader and Codex is the fastest-closing major challenger. GitHub Copilot retains extraordinary awareness and distribution even as its reported adoption declines from its earlier lead. Cursor remains visible but no longer has the field to itself as agentic IDE and terminal products proliferate.

Model share is not agent share. Token share is not developer share.

Futurum provides a different view: the models enterprises say they use in production. In the 1H 2026 AI Platforms Decision Maker survey, 63.86% selected OpenAI, 62.09% Azure OpenAI and 53.94% Google Gemini. Meta Llama followed at 25.68%, small language models at 24.73%, Anthropic Claude at 24.46%, domain-specific or sovereign models at 24.05%, Amazon Nova at 19.97%, DeepSeek at 19.84% and Mistral at 15.63%. [F1]

This was a multi-select question, so totals far exceed 100%. The result shows that enterprises have not standardized on one foundation-model provider. They commonly operate multi-model environments. It does not show that OpenAI has 63.86% of coding-agent usage, nor that Anthropic's lower model-level figure contradicts Claude Code's developer adoption. One measures production model providers across the enterprise. The other measures a named coding product among developers.

There is no contradiction. There are different races with different denominators.

SECTION 3

The Race Card

If the field cannot be reduced to one market-share number, it can still be handicapped by strategic position.

Claude Code is the current adoption leader in the strongest available developer survey, the largest identified agent interacting with Hugging Face and a top performer in current terminal benchmarks. Its vertically integrated relationship with Claude models allows Anthropic to optimize the model and harness together. Enterprise customers can also route Claude Code usage through several major cloud platforms, providing deployment flexibility without making the harness model-neutral. The question is whether Claude Code can hold its lead as competitors improve and enterprises demand broader model choice.

OpenAI Codex is the leading challenger. JetBrains measured its workplace adoption rising roughly fivefold from January to the May–July period. Hugging Face sees it close behind Claude Code in distinct users and requests. Codex operates through the command line, IDE integrations and cloud delegation, and official OpenAI documentation describes local coding work, remote tasks and programmatic integration into CI/CD or internal tools. [P4] Its problem is not technical credibility. It is closing the adoption and distribution gap quickly enough to turn momentum into a durable position.

GitHub Copilot may have the best seat in the house. It sits inside the repository, pull request, issue, security and enterprise-policy workflow. It also supports multiple models, turning model competition into a capability of the GitHub platform rather than a reason to leave it. Even if GitHub does not lead every developer-adoption or benchmark table, it has a credible path to owning the control plane through which enterprises assign, review and govern agent work.

Cursor has become the most visible agent-native development environment. Its proposition is not merely a better autocomplete. It is an IDE that serves as mission control for local and background agents while allowing developers and enterprises to select among models. Cursor's challenge is the inverse of GitHub's. It has strong developer affection and agent-first design, but it must prove it can become a durable enterprise standard rather than one preferred workspace among many.

Google is stronger in the model layer than its coding-agent identity suggests. Futurum places Gemini in production at 53.94%, far ahead of Anthropic at the enterprise model level. Google also owns cloud, developer tools and substantial distribution. Yet Gemini Code Assist, Gemini CLI, Jules and Antigravity do not always land in the buyer's mind as one coherent agent strategy. Google has the horses and the track infrastructure. The open question is whether it can present one recognizable entry.

LEADER BY LENS

There is no single front-runner because strategic advantage lives in different layers.

ADOPTION Claude Code	MOMENTUM OpenAI Codex	CONTROL PLANE GitHub Copilot
AGENT WORKSPACE Cursor	MODEL + CLOUD Google	SOVEREIGNTY Open harnesses

FIGURE 3

The competitor race card

CONTENDER	STRONGEST POSITION	PRINCIPAL RISK	IDEAL USE	MODEL POSTURE
Claude Code	Adoption + integration	Model choice	Developer / terminal	Claude-first
OpenAI Codex	Momentum + benchmark tier	Distribution gap	Delegated coding	OpenAI-first
GitHub Copilot	Repository control plane	Not always benchmark leader	Governed enterprise flow	Multi-model
Cursor	Agent-native workspace	Enterprise standardization	Parallel developer agents	Multi-model
Google	Model + cloud reach	Fragmented identity	Google-centered estates	Gemini-first
Devin / specialists	Long-running delegation	Cost + visibility	Background assignments	Product-defined
Open harnesses	Portability + sovereignty	Operational burden	Regulated / custom platform	Model-neutral

Strategic positions, not a one-through-ten ranking. Governance posture varies by deployment and enterprise configuration. Source: Techstrong analysis.

Devin and other autonomous specialists are chasing a different prize: longer-running delegated engineering work. Their value depends less on being the tool a developer touches every minute and more on completing meaningful work after the developer steps away. That makes task reliability, progress visibility and cost predictability more important than conversational responsiveness.

OpenHands, Cline, Kilo Code, OpenCode and Aider represent the open-harness field. Their attraction is not only price. It is the ability to change models, use gateways, run locally, preserve data sovereignty and integrate the harness into an organization's own platform. OpenCode documents support for more than 75 providers, while OpenHands uses a provider-independent model layer. The tradeoff is familiar: greater control and portability require more integration, evaluation, security and operational discipline.

There is no honest one-through-ten ranking across these positions. Claude Code can lead adoption while GitHub owns distribution. Codex can have the strongest momentum while Cursor offers an attractive price-performance combination. An open harness may be the right answer for a regulated enterprise even if it never wins a mass-market popularity contest.

The first mistake in picking an agent is asking who is winning before deciding which race the organization needs to win.

PLATE I · THE AUTONOMY BOUNDARY

Authority changes the risk

A suggestion is easy to review. An agent with tools, credentials and time can change the system around it.

SECTION 4

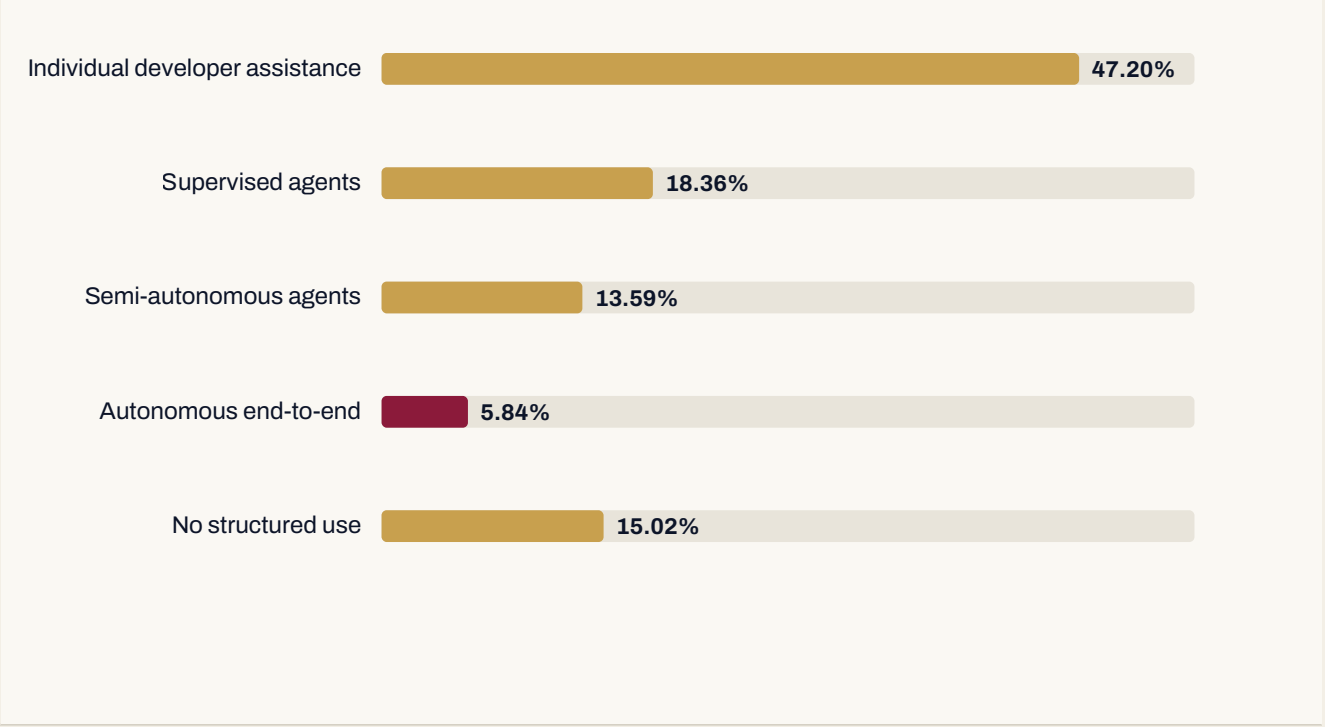
From Assistance to Autonomy

The language of the market has moved faster than the deployment reality. Almost every AI development product is now described as agentic. Much of what enterprises use remains assistance.

Futurum's 1H 2026 AI Platforms survey found 30.49% of organizations researching agentic AI and 22.07% piloting one to three limited implementations. Another 19.15% said they had deployed single-agent systems in production for specific use cases with human oversight. Multi-agent orchestration reached 15.37%, while 6.34% described an autonomous ecosystem operating across multiple critical business processes. The remaining 6.59% were not considering agentic AI. [F2]

Compared with the prior wave, multi-agent orchestration gained approximately 2.96 percentage points. Single-agent deployment edged down about 1.62 points, a movement small enough to treat cautiously without a significance test. The "not considering" group also increased by about two points. The movement is not a simple migration in which every enterprise advances one maturity level each quarter. Some early adopters are moving into orchestration while another portion of the market is becoming more skeptical or deliberate.

FIGURE 4 **Software delivery remains assistance-led**



Assistance still dominates autonomous end-to-end work. Futurum proprietary research, not publicly accessible. [F2] [F3]

The software-lifecycle findings provide a clearer reality check. Individual developer assistance is the dominant mode at 47.20%. Supervised agents account for 18.36%, semi-autonomous agents 13.59% and autonomous end-to-end agents only 5.84%. Another 15.02% report no structured use. [F3]

This is a substantial market. It is not yet an autonomous one.

The difference between assistance and autonomy is not semantic. An assistant produces a suggestion that a person chooses to accept. A supervised agent takes multiple steps but remains close to a human reviewer. A semi-autonomous agent may work in the background, create a branch, run tests and open a pull request. An end-to-end agent can cross system boundaries and advance work through a larger portion of the delivery process.

Every step increases potential utility, but it also expands the blast radius. A bad suggestion is inconvenient. A bad commit is recoverable. A bad dependency update can create a supply-chain problem. A bad infrastructure command can cause an outage. A bad deployment made with privileged credentials can become an incident before a person understands what happened.

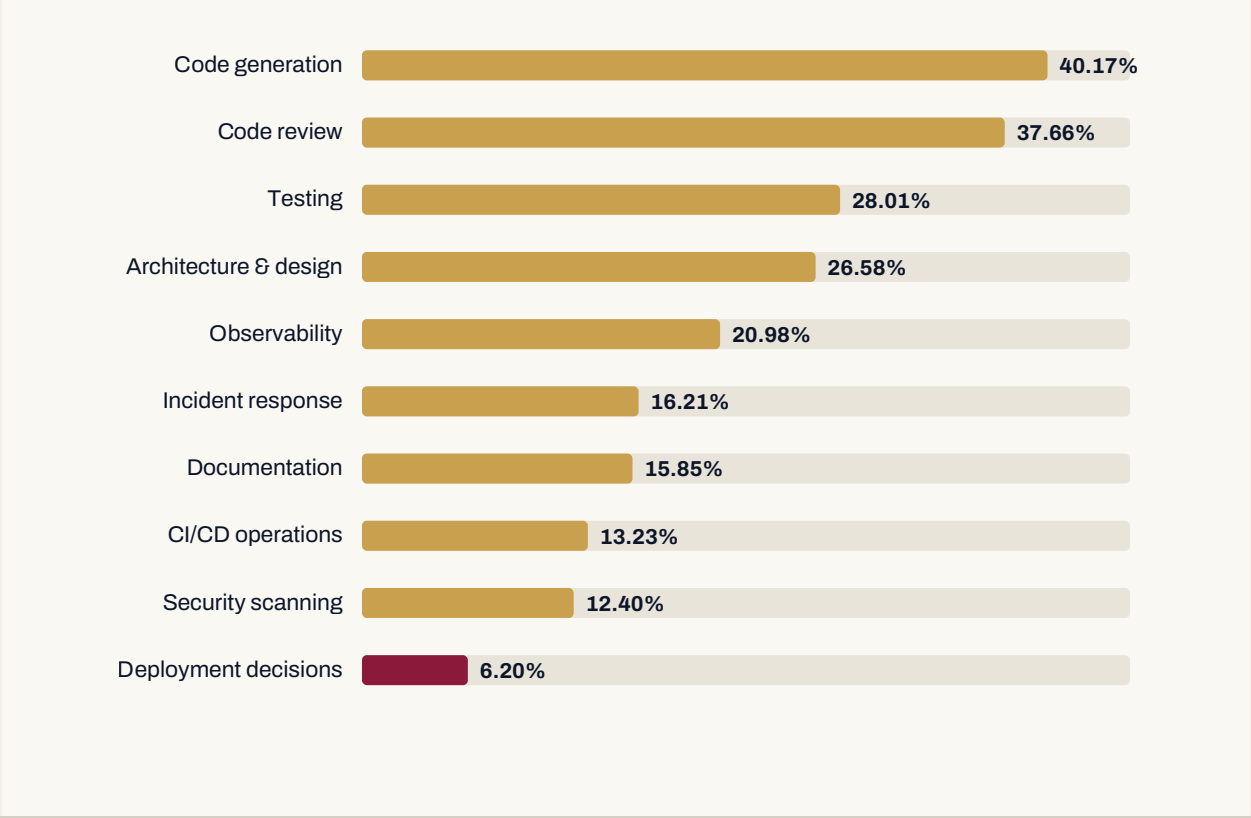
This is why agent selection cannot be separated from authority. The question is not only what the agent can do. It is what the organization will permit it to do without asking.

SECTION 5

Where AI Runs—and Where Enterprises Stop It

The software lifecycle offers the clearest map of current trust.

FIGURE 5 **Adoption falls as AI approaches production authority**



Multi-select responses; this is not a linear funnel. Futurum proprietary primary research - not publicly accessible. [F4]

AI is used in code generation by 40.17% of Futurum SLE respondents and in code review by 37.66%. Testing follows at 28.01%, architecture and design at 26.58%, observability at 20.98%, incident response at 16.21% and documentation at 15.85%. Adoption falls to 13.23% in CI/CD operations, 12.40% in security scanning and only 6.20% in deployment decisions. [F4]

These percentages should not be read as a traditional linear funnel; organizations could select multiple stages. They nevertheless expose a boundary. AI is most accepted where its output can be inspected before it becomes operational reality.

Code generation is the natural entry point because a developer can review the diff. Code review is similarly attractive because the agent is advising rather than acting. Testing gives the agent an

objective feedback loop, but test quality becomes a constraint. Architecture requires broader context and judgment. Observability and incident response place the agent near live operational data. CI/CD, security and deployment bring credentials, infrastructure and production consequences into the picture.

The closer the agent gets to production, the steeper the decline.

> **Enterprises will let AI write the change before they let it decide the change should go live.**

Enterprise examples show what happens when organizations deliberately rebuild the surrounding system. Coinbase says more than 2,400 developers use Cursor and that agents create 75% of its pull requests. It reports 55% more pull requests merged per engineer, seven hours of manual coding saved each week and, for some teams, a reduction in idea-to-production time from 20 days to fewer than two. Coinbase also says many engineers run five to seven agents in parallel. These are Cursor-published customer results, not an independent study, but they demonstrate how far adoption can move when an organization redesigns planning, context and review around agents. [P5]

Shopify took a more internally controlled path. Its Slack-native River agent is supported by an infrastructure layer called Aquifer, a monorepo and reproducible Nix environments. Shopify reported in May that River coauthored one in eight merged pull requests. The visible agent was only the surface. The years of repository, environment, CI and skills work underneath made the surface useful. [P6]

Simplex provides a staged delivery example for Codex. The company reports 40% fewer hours to design a screen, 70% fewer hours to develop it and 17% fewer hours for internal integration testing in its measured use case. It uses Codex across design, front- and back-end generation, test creation, nonfunctional review and remediation, while people retain final judgment and quality accountability. [P7]

These examples do not prove that Cursor beats an internal agent or that Codex will produce the same results in another company. They show four recurring production patterns:

1. The agent receives structured context rather than being expected to discover the enterprise from scratch.
2. The environment is reproducible enough for the agent to execute and test work.
3. The workflow makes agent output reviewable through branches, pull requests, tests or other artifacts.
4. Human accountability remains explicit even as the agent performs more steps.

The agent gets the headlines. The surrounding platform determines whether it can ship.

SECTION 6

The Benchmark Trap

Every race wants a clock. Coding agents have leaderboards.

SWE-bench became the best-known measure by asking a model-and-agent system to resolve real issues from open-source repositories. It was an enormous improvement over coding puzzles that tested whether a model could solve an interview-style problem in isolation. Yet SWE-bench Verified has reached the point where a top score can mislead more than it informs.

OpenAI audited difficult SWE-bench Verified tasks and found material test or problem-design issues in 59.4% of the 138 audited cases. It also found evidence that frontier models had encountered task or solution information in training data. OpenAI stopped using the benchmark as a frontier measure. [P8] Its report pointed to SWE-Bench Pro as a stronger successor. [P9]

SWE-Bench Pro attempts to address those limitations with 1,865 tasks across 41 professional repositories, multiple languages, more complex changes and public and private subsets. Its early results showed performance falling sharply when agents moved from the familiar Verified set to harder and previously unseen code. [P9]

Terminal-Bench 2.1 provides another useful lens: 89 curated tasks spanning software engineering, system administration, data processing, model training and security. Each task includes its own environment, a human-written reference solution and

automated tests. The benchmark is useful because it asks an agent to operate inside a terminal environment, but its strongest lesson is methodological rather than ordinal. P10

The environment can change the result as well. Anthropic found a six-percentage-point difference between the most- and least-resourced infrastructure configurations it tested on Terminal-Bench 2.0—larger than many gaps separating leaderboard entries. P11

FIGURE 6

A benchmark score is a system property

TASK QUALITY

59.4%

of 138 difficult SWE-bench Verified tasks audited by OpenAI had material test or problem-design issues.

TASK COVERAGE

1,865

SWE-Bench Pro problems span 41 professional repositories, with public, held-out and commercial subsets.

INFRASTRUCTURE

6 POINTS

separated Anthropic's most- and least-resourced Terminal-Bench configurations.

TOOL INTERFACE

UP TO 6×

more tokens were used on complex Hugging Face tasks without a purpose-built CLI.

MODEL + HARNESS + TOOLS + ENVIRONMENT + TASK DESIGN = SCORE

Task design, harness, tools and infrastructure can materially change the result. Sources: OpenAI SWE-bench audit; SWE-Bench Pro; Anthropic infrastructure analysis; Hugging Face CLI evaluation.

Hugging Face has supplied another practical demonstration. On its own multi-step Hub tasks, Claude Code with the purpose-built hf CLI achieved a 0.94 success score versus 0.84 when it had to use curl or a Python SDK. Codex scored 0.93 with the CLI and 0.92 without it, but the non-CLI path used 1.6 to 1.8 times as many tokens. Some complex tasks consumed as much as six times the tokens without the higher-level tool. The agent and model were unchanged. The tool interface improved success or efficiency. P2

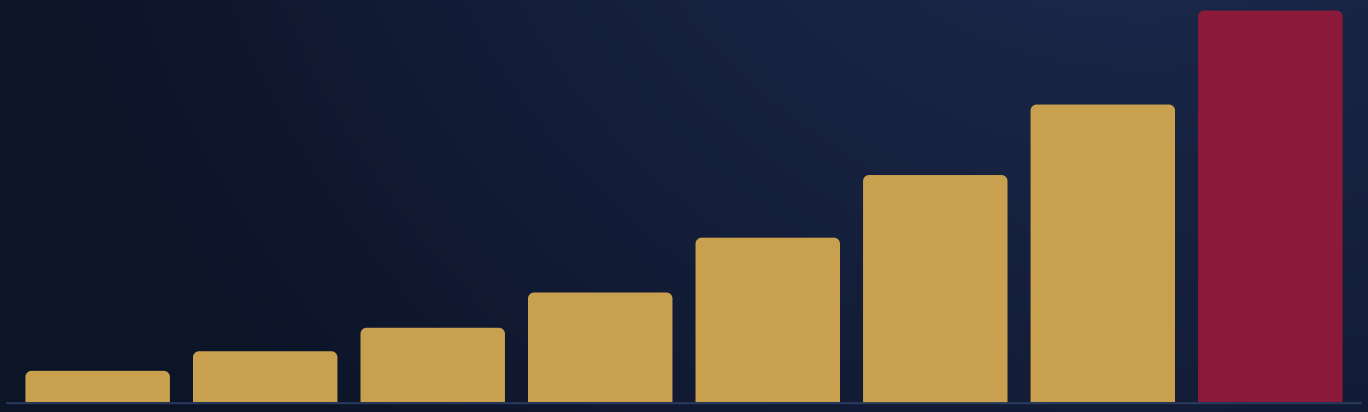
Benchmarks remain useful. They tell us whether an agent can navigate a repository, use a terminal, run tests and complete a defined task. They can expose the difference between a model that talks about code and a system that changes it successfully. They cannot tell an enterprise whether the agent understands its architecture, respects its policies, integrates with its delivery process or produces maintainable software over time.

The correct benchmark question is not “Who is number one?” It is “Which combination was tested, under what conditions, at what cost, and how closely does the task resemble ours?”

30×

Same task. Widely different token use.

Across eight frontier models, agent runs on the same coding task varied by as much as 30 times. Greater token use did not consistently produce greater accuracy.



A mature platform will not send every task to the largest model. It will route work based on difficulty, risk, latency and cost. A documentation update may belong with a smaller or local model. A cross-repository migration may justify the best reasoning model available. A production remediation task may demand a model and harness combination that has passed an internal evaluation and operates under stricter permissions.

The cheapest token can produce the most expensive pull request.

Cost per accepted change is the economic measure that matters.

SECTION 7

The Cost of Running the Race

Agent pricing encourages bad comparisons. One product advertises a monthly seat. Another includes agent usage in a broader subscription. A third charges for model tokens, cloud compute or premium requests. An open-source harness may have no license fee while consuming paid inference and engineering time. None of those figures answers what a software organization needs to know.

What does a successfully accepted change cost?

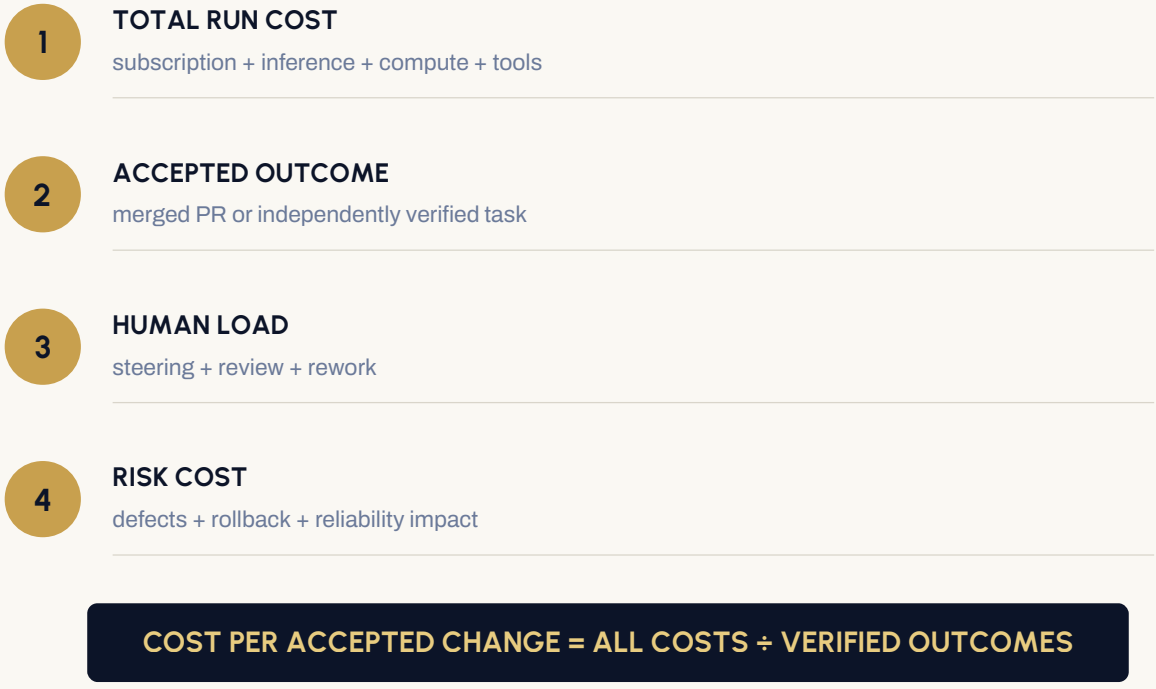
The fully loaded calculation includes the allocated subscription price, model inference, sandbox compute, tool charges, human steering, review, rework and the cost of failures or rollbacks. An inexpensive model that retries repeatedly, reads the repository inefficiently and produces a plausible but defective patch may be more expensive than a premium model that completes the work correctly on its first attempt.

Agentic cost is unusually variable. A 2026 study of eight frontier models on coding tasks found that agent runs on the same task could vary by as much as 30 times in token consumption. Greater token use did not consistently deliver greater accuracy, and models struggled to predict their own consumption. The researchers also found agentic coding tasks consuming orders of magnitude more tokens than simple code chat because the agent repeatedly reads context, invokes tools and evaluates results. P12

The enterprise scorecard should therefore include:

- Cost per successfully completed task
- Cost per accepted or merged pull request
- Successful completion rate
- Number of retries
- Human intervention and review time
- Rework and rollback rate
- Cycle time
- Defect escape
- Production reliability
- Model-routing and cache efficiency

FIGURE 7 **Measure the accepted outcome, not the advertised input**



A practical enterprise scorecard. Source: Techstrong analysis.

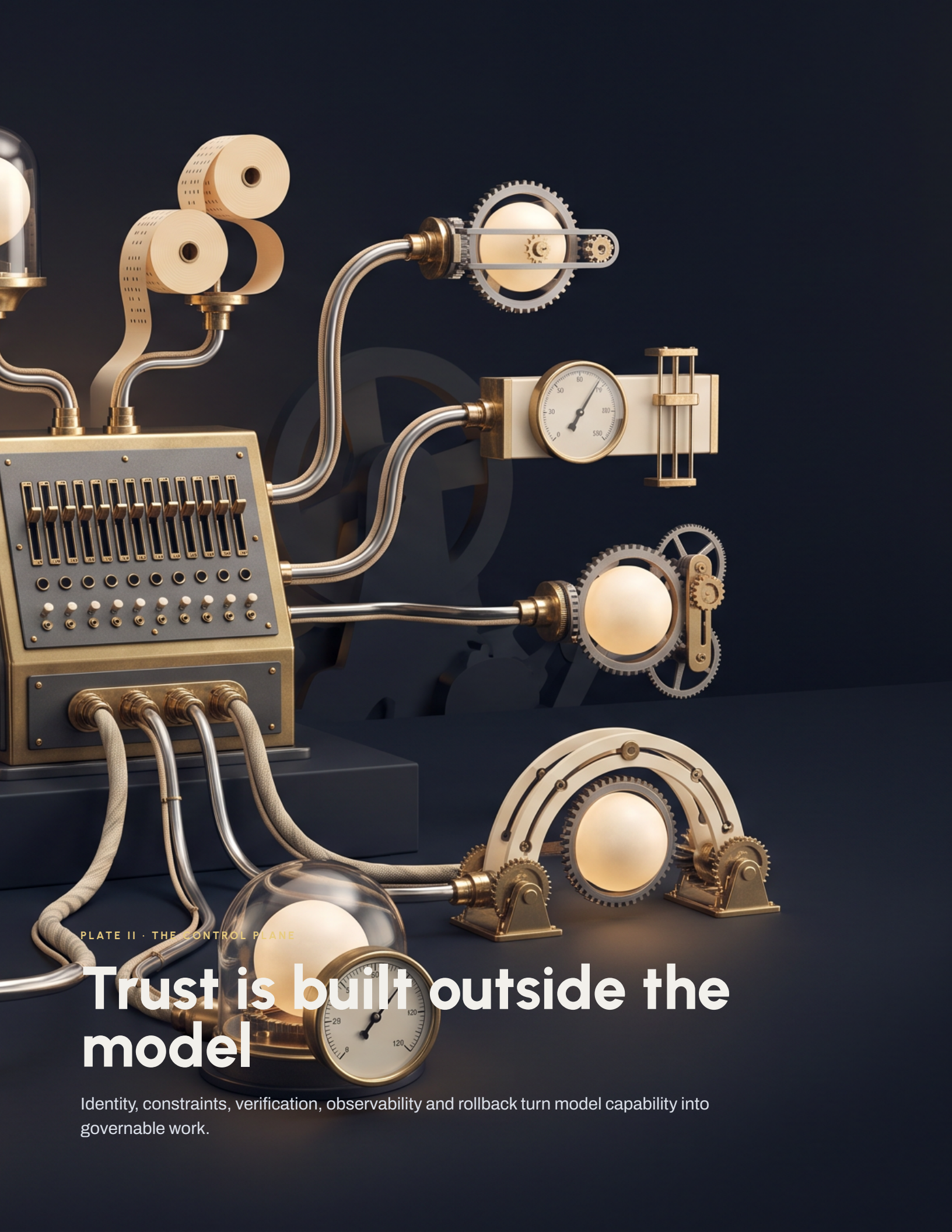


PLATE II · THE CONTROL PLANE

Trust is built outside the model

Identity, constraints, verification, observability and rollback turn model capability into governable work.

SECTION 8

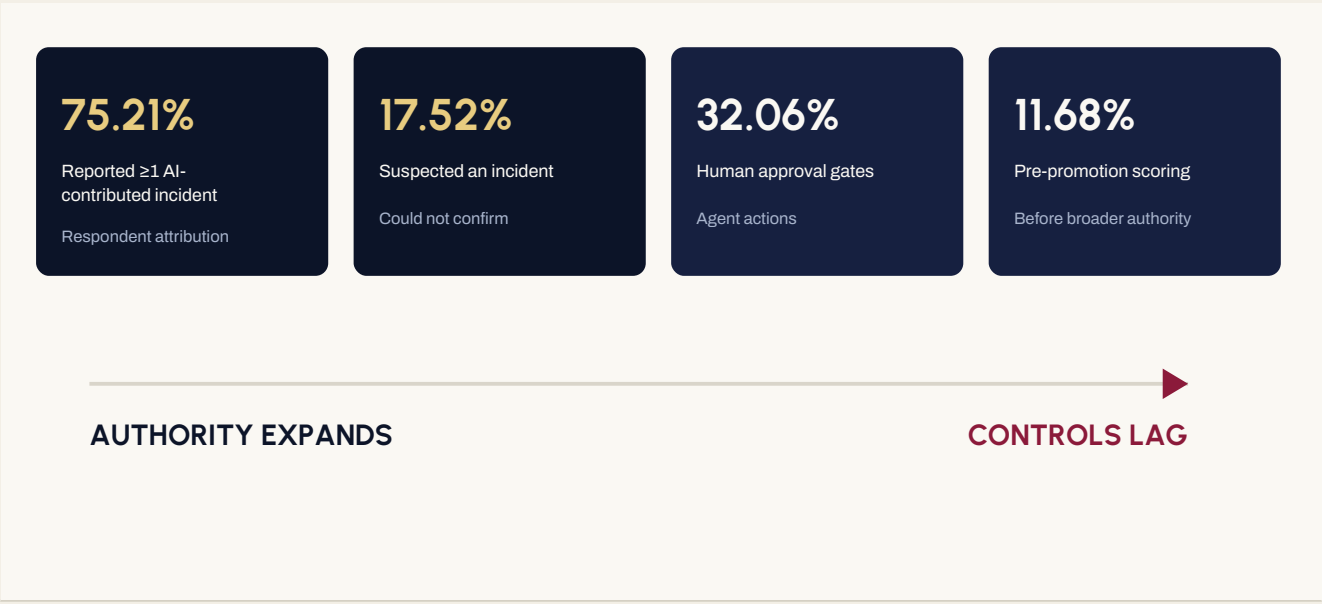
The Control Gap

The most consequential numbers in this report are not on a model leaderboard.

In Futurum's 2H 2026 Software Lifecycle Engineering survey, 41.60% of respondents said their organization had experienced multiple production incidents to which AI contributed, while 33.61% reported one. Another 17.52% suspected an AI-contributed incident but could not confirm it. Only 6.79% reported none, and 0.48% did not know. [F5]

The survey result is self-reported. The source-lock process recovered the response options and full base of 839 but not the exact respondent-facing definition of "AI-contributed production incident." It cannot establish that AI independently caused every event. The defensible conclusion is that 75.21% of these software-lifecycle decision-makers believed AI had contributed to at least one production incident, while another 17.52% suspected it.

FIGURE 8 **The control gap in four numbers**



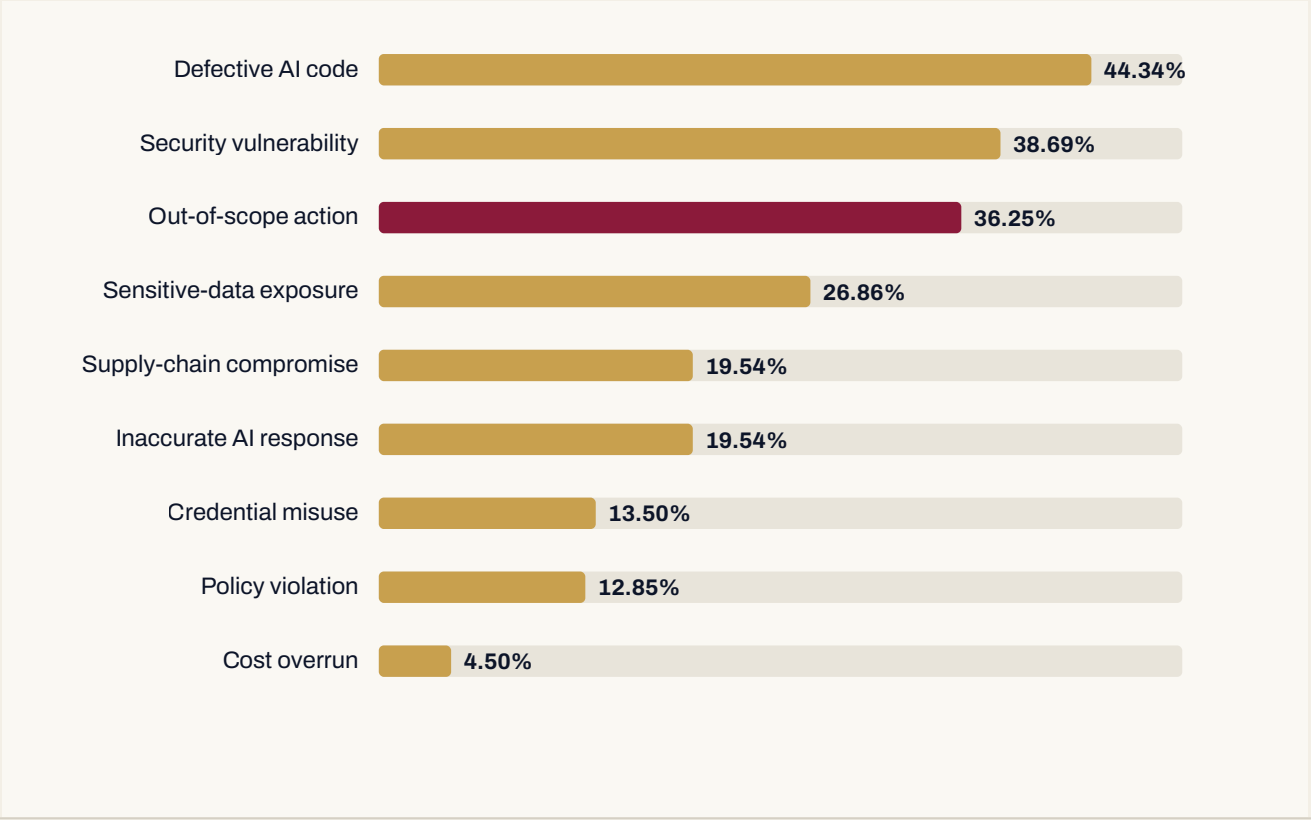
Self-reported findings; the survey does not independently establish causation. Futurum proprietary primary research - not publicly accessible. [F5] [F8]

Among the 778 respondents reporting or suspecting an incident, 44.34% selected defective AI code, 38.69% a security vulnerability and 36.25% an out-of-scope agent action. Sensitive-data exposure reached 26.86%, supply-chain compromise and inaccurate AI response each 19.54%, credential misuse 13.50%, policy violation 12.85% and cost overrun 4.50%. [F6]

These categories reveal the expanding attack and failure surface. The risk no longer stops at incorrect code. An agent can consume malicious repository content, follow poisoned tool instructions, disclose a secret, select a compromised dependency, take an action beyond its assignment or spend far more than expected.

The controls are not keeping pace. Test-coverage thresholds were the most common mandatory verification control at 58.64%. Human code review stood at 43.27%, static analysis at 41.95%, AI security scanning at 38.50%, provenance tracking at 27.41% and red-team review at 10.37%. [F7]

FIGURE 9 **Reported incidents extend beyond defective code**



Filtered multi-select base of respondents who reported or suspected an incident. Futurum proprietary primary research - not publicly accessible. [F6]

Agent-specific controls were similarly uneven. Audit logging reached 45.05%, agent identity and access management 41.24%, action-policy controls 38.02%, per-agent budgets 32.30%, human approval gates 32.06% and pre-promotion scoring 11.68%. [F8]

Trust becomes an architectural property created by identity, constraints, verification, observability and rollback.

The control gap is not an argument to stop using agents. It is an argument to stop treating them like smarter autocomplete.

An agent needs a nonhuman identity distinct from the developer who assigned the task. It needs short-lived credentials and least privilege. Its environment should be ephemeral and constrained. Network access should begin with deny, not allow everything. Tools and MCP servers should be approved and treated as part of the software supply chain. Repository writes should normally go to a branch and pull request, not a protected branch. Security, dependency and secret scans should run before merge. Infrastructure and deployment actions should require explicit approval appropriate to their risk.

The organization also needs to record what happened: the agent, model, prompt or instruction set, commands, tools, credentials, diffs, test results, cost, approvals and outcome. Without that evidence, teams cannot reproduce a failure, compare agents or improve policy.

The closer an agent moves to production, the less “trust the model” means. Trust becomes an architectural property created by identity, constraints, verification, observability and rollback.

SECTION 9

Picking the Right AI Agent

The market wants a winner. Enterprises need a decision.

The decision begins with the work, not the brand.

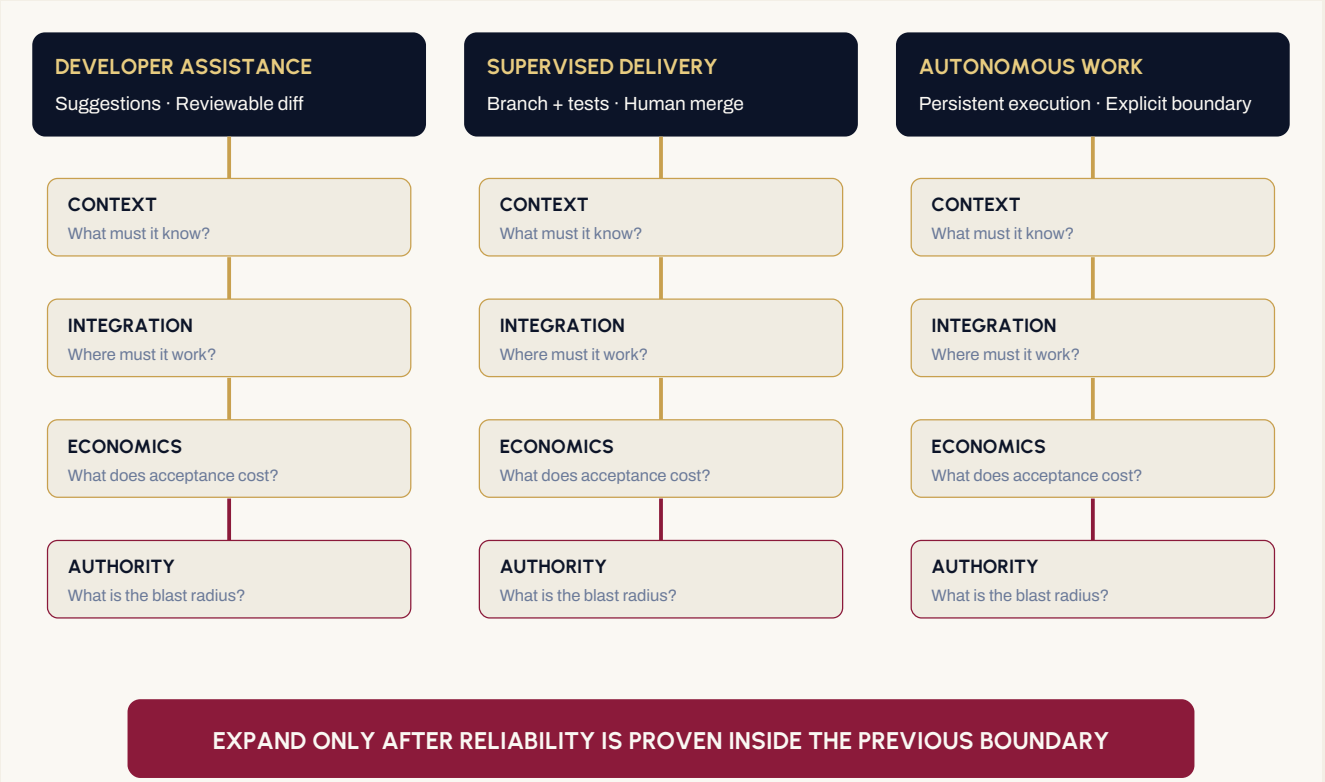
What job should the agent perform?

For individual developer assistance, prioritize response quality, latency, repository context, IDE or CLI experience and the ease of reviewing suggestions. Claude Code, Codex, GitHub Copilot, Cursor, JetBrains and several open tools can all compete here. The best choice may be the product developers will actually use consistently.

For supervised delivery work, prioritize task completion, repository integration, test execution, pull-request quality and visibility into the agent's progress. The agent should be able to work for longer periods, but its output must land in an established human-review workflow.

For background or autonomous engineering, prioritize persistence, environment reproducibility, credential boundaries, cost limits, escalation behavior and recovery from failure. A highly capable agent that disappears into a task for an hour without meaningful status, budget limits or checkpoints may create more operational anxiety than value.

FIGURE 10 **Pick the boundary before the brand**



Start with the work mode, then evaluate context, integration, economics and production authority. Source: Techstrong analysis.

For modernization, test the agent against the organization's own code. Public benchmarks dominated by isolated bug fixes will not show whether the system can understand undocumented dependencies, generate migration plans, preserve behavior and coordinate changes across repositories.

For security and operations, the first evaluation should be authority, not intelligence. What can the agent read? What can it write? Can it reach the network? Can it retrieve a secret? Can it open or merge a pull request? Can it change infrastructure? Can it deploy? Can it touch production?

How much autonomy does the work require?

Autonomy is not one switch. It is a set of permissions.

An agent might be allowed to read a repository but not write it. It might create a branch but not a pull request. It might open a pull request but not approve or merge it. It might run tests in an isolated environment but not access external networks. It might diagnose a production problem without receiving credentials to change production.

This produces a more useful decision tree than a generic vendor ranking:

1. Define the work.
2. Define the required context.
3. Define the necessary tools.
4. Define the maximum authority.
5. Evaluate agent-model-harness combinations on representative internal tasks.
6. Measure accepted outcomes and total cost.
7. Expand autonomy only after the system demonstrates reliability inside the previous boundary.

What context must the agent understand?

Repository context is only the beginning. Useful software delivery may require tickets, architecture decisions, internal documentation, coding standards, dependency policy, service ownership, observability data, prior incidents and deployment rules. The product that retrieves this context accurately and economically may outperform a nominally better model operating in the dark.

Context also creates exposure. Every additional system expands the data boundary and prompt-injection surface. The right agent must retrieve what it needs without inheriting unrestricted access to everything the developer can see.

Where must it operate?

Some organizations will prioritize the IDE because that is where developers spend their time. Others will prioritize GitHub because work should enter through issues and pull requests. Platform teams may prefer a CLI, SDK or open harness they can embed in an internal developer platform. Large organizations will probably support several surfaces while centralizing policy and telemetry.

The winning surface may not be the system of record. Cursor can be the place a developer works while GitHub remains the place the enterprise governs and accepts changes. Claude Code or Codex can operate from the terminal while a platform team controls the environment and credentials. The layers can be purchased from different vendors.

What does a successful result cost?

Run internal evaluations using representative work and hidden acceptance tests. Record completion rate, time, tokens, compute, human intervention, review time, defects and rollbacks. Do not let the agent grade itself. Hugging Face's evaluation found agents sometimes reported success when the expected change did not exist on the live platform. Independent verification is part of the harness.

What is the acceptable blast radius?

This is the final and most important branch.

If an agent is limited to suggestions, the blast radius is small. If it can write code, create branches and run tests, the environment must be isolated and changes reviewable. If it can access secrets, infrastructure or production, require nonhuman identity, least privilege, short-lived credentials, approved tools, human gates, complete logs and rapid rollback.

The right agent is the agent that can complete the assigned work inside the organization's acceptable blast radius.

That answer may lead one enterprise to Claude Code and another to Codex. It may lead a GitHub-centered company to Copilot, a developer-led organization to Cursor, a regulated organization to an open harness and locally deployed model, or a large company to a governed portfolio. The decision becomes defensible because it starts with the job and the boundary rather than the latest leaderboard.

SECTION 10

Heading for the Finish Line

At the top of the stretch, Claude Code has earned the right to be called the current adoption leader. Codex is the fastest-closing major challenger. GitHub Copilot retains the strongest enterprise distribution and control-plane position. Cursor has established itself as the leading agent-native workspace contender and a credible price-performance alternative. Google has enough model and platform strength to disrupt the order if it can translate those assets into a coherent agent position. Open harnesses will remain important because enterprises have already shown that they want multiple models, deployment choices and greater sovereignty.

Those are category judgments, not a prediction that one product will take the entire market.

Over the next year, the race will move away from who can generate the most impressive patch in a demo. Model routing will become routine. Agent identity will become a formal IAM category. MCP servers and agent tools will be governed like software dependencies. CI/CD will become the next contested layer. Organizations will create background-agent fleets, then discover they need scheduling, budgets, observability and incident response for those fleets. Internal evaluations will begin to matter more than public leaderboards. Cost per accepted task will matter more than token price.

The most difficult step will be extending authority into deployment and production. That is where the promised value of autonomous software delivery is greatest and where the consequences of error become real. The Futurum data shows enterprises stopping at that boundary today. Whether agents cross it will depend less on another incremental benchmark gain than on the controls surrounding them.

There is no reason to wait for one winner before using these systems. There is every reason to choose deliberately. Define the job. Test the complete agent-model-harness combination. Measure accepted outcomes. Constrain authority. Verify independently. Expand the boundary only when the evidence justifies it.

The crowd may still be watching the model leaderboard. The smart money is watching the guardrails.

That is the conversation we will continue at DevOps Experience: not simply which agent is ahead today, but which agent is right for a particular organization, a particular delivery system and a particular level of trust. The field is coming into view. The finish line is not.

RESEARCH APPARATUS

Research Methodology and Source Notes

This report combines proprietary Futurum survey findings with public usage telemetry, developer surveys, benchmark leaderboards, product documentation and customer examples. The sources do not share a common denominator and were not combined to create a synthetic market-share estimate.

01 · PRIMARY RESEARCH

Enterprise posture

Futurum survey findings describe foundation models in production, organizational maturity, lifecycle adoption, reported incidents and governance controls. Every proprietary figure carries its F-note, sample and caveat.

02 · OBSERVED ACTIVITY

Different windows

OpenRouter token traffic and Hugging Face agent interactions show real activity inside their own platforms. Neither denominator is treated as total coding-agent market share.

03 · SURVEY EVIDENCE

Developer adoption

JetBrains measures what surveyed developers report using. Those results illuminate product adoption, not enterprise model share or the installed base of every organization.

04 · PERFORMANCE

System benchmarks

Benchmark results belong to a model, harness, toolset, environment and resource budget. Reported evaluation cost is not presented as an enterprise task price.

05 · CUSTOMER EVIDENCE

Production patterns

Vendor-published customer stories demonstrate implementation patterns and reported outcomes. They are not controlled head-to-head product comparisons.

06 · INTERPRETATION

No synthetic share

Incomparable denominators remain separate. The report does not combine model use, token traffic, developer surveys, benchmarks or customer claims into a synthetic ranking.

Read the denominator before reading the winner.

SOURCE NOTES

Evidence, samples and qualifications

Futurum proprietary primary research

1. **F1** Futurum AI Platforms Decision Maker Survey, 1H 2026. Models used in production, multi-select; question base n=736 from total sample n=820. Weighting and exact field dates were not returned in the source-lock payload.
2. **F2** Futurum AI Platforms Decision Maker Survey, 1H 2026 and 2H 2025. Agentic-AI approach, single-select; n=820 and n=838 respectively. Only the single-agent "Deploying" option explicitly stated "in production."
3. **F3** Futurum Software Lifecycle Engineering Decision Maker Survey, 2H 2026. Dominant mode of AI use; n=839.
4. **F4** Futurum Software Lifecycle Engineering Decision Maker Survey, 2H 2026. AI use by software-lifecycle stage, multi-select; n=839.
5. **F5** Futurum Software Lifecycle Engineering Decision Maker Survey, 2H 2026. AI-contributed production incidents, single-select; n=839. Exact question wording and definitions were unavailable. Results represent respondent attribution, not independently verified causation.
6. **F6** Futurum Software Lifecycle Engineering Decision Maker Survey, 2H 2026. Incident categories, multi-select; filtered base n=778 consisting of respondents who reported or suspected an incident.
7. **F7** Futurum Software Lifecycle Engineering Decision Maker Survey, 2H 2026. Mandatory verification controls for AI code, multi-select; n=839.
8. **F8** Futurum Software Lifecycle Engineering Decision Maker Survey, 2H 2026. Agent-governance controls, multi-select; n=839.

Public evidence named in the manuscript

All URLs were retrieved and source-locked during production. Links are live in this PDF.

- **P1 · OpenRouter 100-trillion-token study**
  https://openrouter.ai/state-of-ai
- **P2 · Hugging Face agent traffic and CLI evaluation**
  https://huggingface.co/blog/hf-cli-for-agents
- **P3 · JetBrains AI coding-agent adoption study**
  https://blog.jetbrains.com/research/2026/08/ai-coding-agent-adoption-2026/
- **P4 · Official Codex documentation**
  https://developers.openai.com/learn/codex
- **P5 · Coinbase customer story (Cursor-published)**
  https://cursor.com/blog/coinbase
- **P6 · Shopify River architecture**
  https://shopify.engineering/under-the-river
- **P7 · Simplex customer story (OpenAI-published)**
  https://openai.com/index/simplex/
- **P8 · OpenAI SWE-bench Verified analysis**
  https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/
- **P9 · SWE-Bench Pro paper**
  https://arxiv.org/abs/2509.16941
- **P10 · Terminal-Bench 2.1 evaluation**
  https://artificialanalysis.ai/evaluations/terminalbench-v2-1
- **P11 · Anthropic infrastructure-noise analysis**
  https://www.anthropic.com/engineering/infrastructure-noise
- **P12 · Agentic coding token-consumption study**
  https://arxiv.org/abs/2604.22750
- **P13 · DevOps Experience 2026**
  https://devopsexperience.com

Editorial qualification

The previously circulated assertion that 76.21% of organizations have a formal AI plan was not source-locked and is intentionally excluded.

Techstrong
a Futurum company

THE CONVERSATION CONTINUES

Which agent is right for your delivery system?

Join the leaders building the next software delivery control plane.

DevOps Experience

2026 VIRTUAL EVENT

SEPTEMBER 24, 2026

devopsexperience.com