/thoughtworks

Design. Engineering. AI.



Technology Radar

An opinionated guide to today's technology landscape

Volume 35
November 2026

About the Radar 3
Radar at a glance 4
Contributors 5
Production credits 6
Themes 8
The Radar 11
Techniques 14
Platforms 29
Tools 38
Languages and Frameworks 50

About the Radar

Thoughtworkers are passionate about technology. We build it, research it, test it, open source it, write about it and constantly aim to improve it — for everyone. Our mission is to champion software excellence and revolutionize IT. We create and share the Thoughtworks Technology Radar in support of that mission. The Thoughtworks Technology Advisory Board, a group of senior technology leaders at Thoughtworks, creates the Radar. They meet regularly to discuss the global technology strategy for Thoughtworks and the technology trends that significantly impact our industry.

The Radar captures the output of the Technology Advisory Board's discussions in a format that provides value to a wide range of stakeholders, from developers to CTOs. The content is intended as a concise summary.

We encourage you to explore these technologies. The Radar is graphical in nature, grouping items into techniques, tools, platforms and languages and frameworks. When Radar items could appear in multiple quadrants, we chose the one that seemed most appropriate. We further group these items in four rings to reflect our current position on them.

For more background on the Radar, see thoughtworks.com/radar/faq.

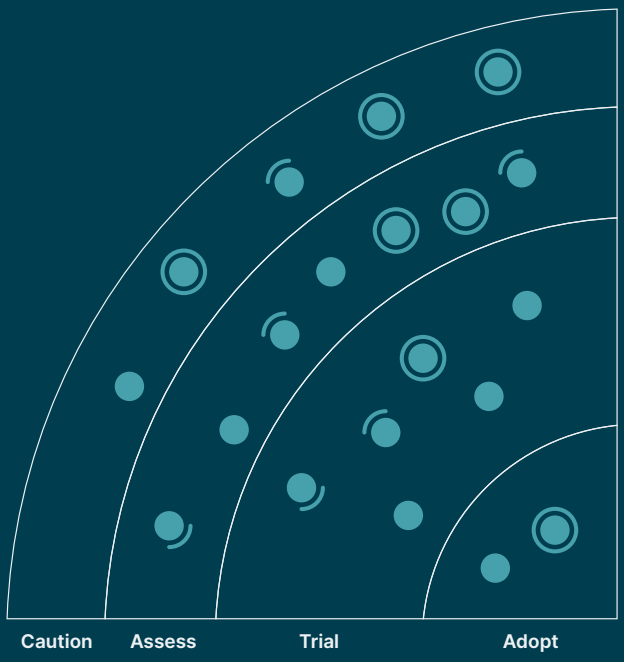


© Thoughtworks, Inc. All Rights Reserved.

Radar at a glance

The Radar is all about tracking interesting things, which we refer to as blips. We organize the blips in the Radar using two categorizing elements: quadrants and rings. The quadrants represent different kinds of blips. The rings indicate our recommendation for using that technology.

A blip is a technology or technique that plays a role in software development. Blips are “in motion”, which is why their position in the Radar often changes. This usually indicates our increasing confidence in recommending them as they move through the rings.



Adopt: We feel strongly that the industry should be adopting these items. We use them when appropriate in our projects.

Trial: Worth pursuing. It’s important to understand how to build up this capability. Enterprises can try this technology on a project that can handle the risk.

Assess: Worth exploring with the goal of understanding how it will affect your enterprise.

Caution: Proceed with care. These items involve significant concerns that require careful evaluation for your specific context.

New **Moved in/out** **No change**

Our Radar is forward-looking. To make room for new items, we fade items that haven’t moved recently, which isn’t a reflection on their value but rather on our limited Radar real estate.

© Thoughtworks, Inc. All Rights Reserved.

Contributors

The Technology Advisory Board (TAB) is a group of 18 senior technologists at Thoughtworks. The TAB meets twice a year face-to-face and biweekly virtually. Its primary role is to be an advisory group for Thoughtworks CTO Rachel Laycock.

The TAB acts as a broad body that can look at topics that affect technology and technologists at Thoughtworks. This edition of the Thoughtworks Technology Radar is based on a meeting of the TAB in Recife, Brazil in September 2026.



Rachel Laycock (CTO)



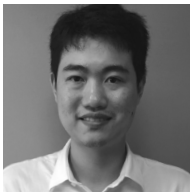
Alessio Ferri



Brandon Cook



Cecilia Geraldo



Chris Chakrit Riddhagni



Chris Ford



Effy Elden



Kief Morris



Muhilvarnan V



Ken Mugrage



Nati Rivera



Ni Wang



Renan Martins



Selvakumar Natesan



Shangqi Liu



Thiago Gregorio



Vanitha Kumar



Vanya Seth

© Thoughtworks, Inc. All Rights Reserved.

Production credits

Editorial team

- **Ken Mugrage**
  Technical Editor
- **Nati Rivera**
  Product Manager
- **Debashri Dutta**
  Project and Campaign Manager
- **Osama Shaikh**
  Project and Campaign Manager
- **Richard Gall**
  Content Editor
- **Kriti Anand**
  Content Editor
- **Michael Koch**
  Copy Editor
- **Gareth Morgan**
  Head of Content and Thought Leadership

Digital and web experience

- **Rashmi Naganur**
  Business Analyst
- **Brigitte Britten-Kelly**
  Digital Content Strategist
- **Vandita Kamboj**
  UX Designer
- **Lohith Amruthappa**
  Analytics Specialist
- **Mukilan Kesavan**
  Developer Lead

Marketing infrastructure

- **Sahil Gupta**
  Marketing Automation Specialist

Design and multimedia

- **Leticia Nunes**
  Visual Designer
- **Sruba Deb**
  Visual Designer
- **Kevin Barry**
  Multimedia Specialist
- **Ryan Cambage**
  Multimedia Specialist
- **Anish Thomas**
  Multimedia Designer

Communications

- **Shalini Jagadish**
  Internal Communications Specialist
- **Yasmin Brussulo**
  Internal Communications Specialist
- **Abhishek Kasegaonkar**
  Social Media Specialist
- **Marsh Abraham**
  Public Relations Specialist
- **Anushree Tapuriah**
  Campaigns and Advertisement Specialist

Tech Radar Development team

- **Nati Rivera**
  Product Manager
- **Alejandro Batanero**
  Technical Lead
- **Raluca Iordache**
  Developer
- **Nacho Martínez**
  Developer

© Thoughtworks, Inc. All Rights Reserved.

Production credits

Thoughtworkers contributions

India

- Abdul Jeelani
- Akash Dixit
- Akshar Srivastava
- Anil EM
- Anitha Lakshmi V
- Ankit Adlakha
- Ankita Verma
- Anusha Panyala
- Balaji G Sunil Kumar
- Birlla P
- Chetan
- Daniel Euchar
- Dhananjay Shelke
- Deepthi R
- Jonathan Cole
- Karthikeyan Vijayan
- Krishnaswamy Subramanian
- Kumar Sankara Iyer
- Lucky Bajaj
- Maheshkumar Kharade
- Malini S
- Mayank Jain
- Mayank Kumar
- Nikhil Mane
- Nithya Sivakumar
- Pallavi Sinha
- Partha Dash
- Piyush Bhargava
- Pranesh Pokale
- Pritam Kumar
- Raghav Jain
- Rahul Garg
- Rajasekaran Gopal
- Rajat Lakhina
- Saiprabhanj Turaga
- Sakthe Karthika
- Sarath Kumar
- Saurabh Machave
- Shadayani Varshney
- Shanmugapriya Srinivasan
- Stevin Sony
- Sudhir Yelikar
- Suman Maity
- Sunit Parekh
- Sureshkumar V
- Vinoth Kumar

Europe

- Abel Guillen
- Alejandro Batanero
- Alejandro Torres Hernández
- Andrew Harmel-Law
- Arif Wider
- Ayhan Tasyurt
- David Rubio
- Gaurav Patole
- Ivan Vidal
- Jose (Pepe) S. Gomez de Olea
- Kiran Prakash
- Luca Giuberti
- Marco Baroetto Parisi
- Martin Kramer
- Matteo Vaccari
- Moritz Wilke
- Otto Tasnadi
- Raquel Devesa
- Ricardo Piccoli
- Rieke Heinze
- Scott Davies
- Sergio Jimenez
- Shweta Justice
- Tamara Carpintero
- Yuvaraj Birari

APAC

- Bojian Chen
- Guangyi Li
- Siyuan Li
- Xiaoming Gao
- Yijie
- Amit Singh
- Bank Thanayut Tiratatri
- Kaiwanyawit Chawankul
- Kelvin Nicholson
- Lilly Ryan
- Manish Joshi
- Michael Joyce
- Naaman Campbell
- Natthapong Intharak
- Neil Price
- Nguyen Viet Son
- Noor Yeaser Khan
- Rose Aunaetitrakul
- Salah Chalermthai
- Santi Liwsakphaiboon
- Thomas Hoang

Americas

- Aravind Kunnath
- Daniel Marinho
- Daniel Romero
- Daniel Santibañez
- Danilo Ferreira
- Enderson Cândido
- Gustavo Rivas
- Jipson Medardo Saad
- Loreto Sanchez
- Luiz Miccieli
- Marco Vela
- Maria Augusta Mateus
- Maria Rosa
- Milton Oliveira
- Nicolas Baquero
- Pablo Sosa
- Rocio Mera
- Sebastián Santiago
- Sebastián Stuardo
- Sergio Jara
- Varun V M
- Victoria Souza
- Yani Ovando

© Thoughtworks, Inc. All Rights Reserved.

Themes



Harness engineering at scale

There was debate about whether to include harness engineering as an individual blip in the Radar, but given it's a theme that connects many techniques and technologies, we decided to highlight it as such in this volume.

Although the term isn't new, this edition highlights a growing ecosystem of practices and tools that shows how harness engineering (everything we do to construct the environment around an AI agent, from memory and context to feedback loops and guardrails) is maturing into a more cohesive discipline. In particular, an emerging plugin ecosystem is taking shape, illustrated by Agent Plugins, agent plugin marketplace and supporting tools such as Agent Package Manager.

Alongside these developments, it's becoming clear that the harness is a living asset that requires ongoing ownership and maintenance. We've discussed, for example, harness quality evaluation, building a self-improving harness and using a feedback flywheel to ensure agents improve over time. The broader challenge is scaling the harness to meet enterprise needs. In the coming months, we may see the enterprise harness emerge as a product, not unlike what happened with platforms and platform engineering a few years ago.

Beyond frontier models

We're entering a new phase of AI model use, one where automatically reaching for a proprietary frontier model is no longer the obvious choice. This edition highlights significant innovation in this space, encouraging organizations to think more creatively about which models to use, how and when.

Open-weight models are particularly notable here. Twelve months ago, the gap with proprietary frontier models was stark, but today it's much smaller for many tasks. Cost is a major driver, but not the only one: open-weight models can run in your public cloud environments, on neoclouds or on your own hardware. That flexibility can help meet strict privacy requirements and reduce dependence on a single vendor. This is especially visible in software engineering, where the GLM family has made open-weight models credible alternatives to proprietary frontier models for some coding-agent workloads. Self-hosting can also help organizations keep source code and prompts within their own infrastructure. But the shift isn't limited to open-weight models. Local coding models are rising in popularity, while developments such as Foundation models for structured data and decision models like Jev reflect growing recognition that frontier LLMs aren't the only approach to AI.

© Thoughtworks, Inc. All Rights Reserved.

Themes



With this growing range of options, choosing the right model for each task is becoming an important responsibility for engineering teams. Model routing is emerging as a key consideration. We caution against blanket frontier model usage and encourage teams to make deliberate model choices while remaining open to alternatives.

Putting more agents to work

In our previous edition, coding agents drove most of our blips and themes. This time, many proposals focused on putting more agents to work, mostly describing developers running several agent sessions in parallel and steering each one. Others described handing agents larger pieces of work that run for longer, such as unattended overnight agentic builds, where developers define the tasks and review the results. This is a lighter-weight approach than the coordinated multi-agent approaches we've covered in previous editions. A team of coding agents assigns agents roles in a predefined workflow. Coding agent swarms go further, with orchestrating agents dividing and assigning work themselves. In both approaches, developers delegate more decision-making. Thoughtworkers are mostly running multiple sessions under more direct control, which we think is a sensible place for engineers to start.

Of the many tools we discussed for managing parallel sessions, including Claude Squad, T3 Code, Orca and Paseo, Herdr attracted the most interest. Most include an attention inbox that organizes sessions by status and prioritizes those waiting for an answer or approval. Omnigent has one, and Paseo lets developers check sessions from their phone. Git worktrees give each of these sessions its own working copy, although they can leave behind a tangle of partially merged branches. Within a session, subagents take on narrow tasks and keep exploration and retries out of the main context. AgentsView lets developers review past sessions, while Hunk helps them review agent-generated changes.

As developers run more agents, their attention and review capacity become bottlenecks. When review can't keep up, oversight risks becoming human-in-the-loop theater. The pressure extends downstream. When several developers each run multiple agents, changes can arrive faster than the delivery pipeline can handle them. Path-to-production mapping helps identify where work queues up. Before adding sessions, teams should consider how much their review and delivery processes can handle. In the near term, we expect more practices and tools to emerge for orchestrating agentic work across multiple engineers. Tools such as Entire CLI, which shares agent sessions across a team, are an early sign of growing interest in making multi-engineer agentic work more effective.

© Thoughtworks, Inc. All Rights Reserved.

Themes



Constructing the AI factory

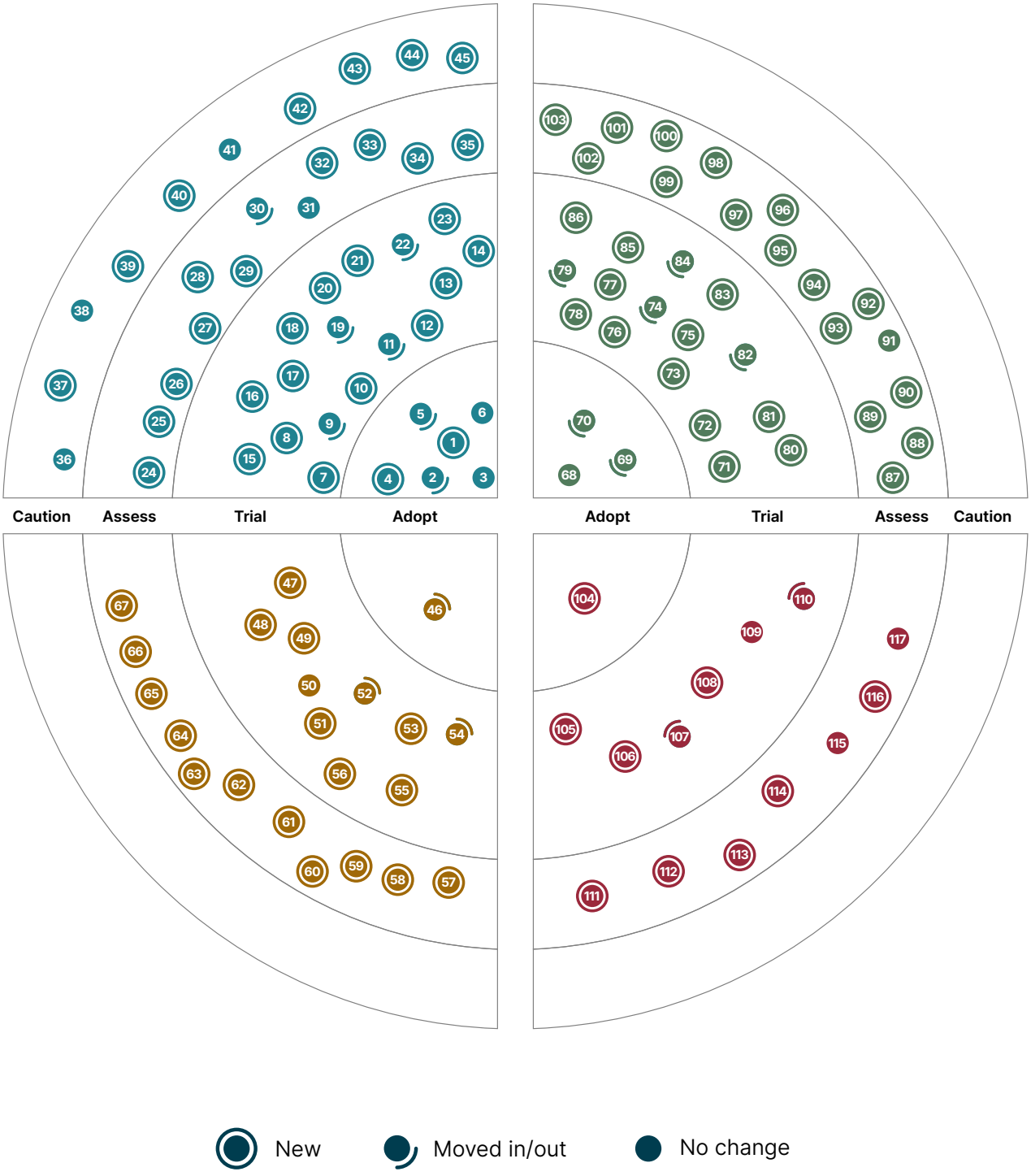
While the demands of AI infrastructure have appeared in several recent Radar volumes, this edition brings them into much sharper focus. As generative AI and agent use intensify, enterprises are discovering that token economics ultimately comes down to infrastructure economics. The cost of AI depends not only on model pricing but also on model selection, routing, batching, data movement and the efficiency of the underlying platform. Much of the innovation needed to make AI economically sustainable therefore sits deeper in the stack and often falls below many people's radar.

In putting this edition together, we encountered a vibrant ecosystem emerging around AI infrastructure. Kubernetes, in particular, is becoming a key foundation for what are sometimes described as AI factories: industrialized platforms for running AI workloads at scale. Several blips illustrate this shift. KServe and KAITO support model serving, deployment and accelerator-aware workload management. Higher in the stack, llm-d tackles distributed inference and scheduling, while LiteLLM provides routing and abstraction across models and inference providers. The AI factory stack also extends beyond compute to storage and networking. Technologies such as VAST DataStore address the high-throughput storage demands of accelerator-heavy workloads, while Netris brings software-defined network automation to the underlying infrastructure, helping teams provision and operate the high-bandwidth connectivity AI clusters depend on.

The bigger shift is that models, GPUs, inference engines, storage and networks are increasingly managed as programmable resources coordinated through a common platform layer, making AI infrastructure a central factor in enterprise AI economics.

© Thoughtworks, Inc. All Rights Reserved.

The Radar



© Thoughtworks, Inc. All Rights Reserved.

Techniques

Adopt

1. Agent sandboxing
2. Mutation testing
3. Path-to-production mapping
4. Post-quantum cryptography
5. Progressive context disclosure
6. Threat modeling

Trial

7. Agent plugin marketplace
8. Agent-maintained knowledge bases
9. Anchoring coding agents to a reference application
10. Coding harness for general-purpose agents
11. Feedback flywheel
12. Graph-grounded codebase reasoning
13. Grilling
14. Interactive visual plans with agents
15. Managed agents runtime
16. Open-weight models
17. Risk-tiered human oversight
18. Self-improving harness
19. Small language models
20. Stateful guardrails
21. Subagents for better context management
22. Toxic flow analysis for AI
23. Unattended overnight agentic builds

Assess

24. Agent pipeline responsibility
25. Agent suggestion box
26. Context profiling
27. Enterprise harness
28. Foundation models for structured data
29. Harness quality evaluation
30. Local coding models
31. MITRE ATLAS
32. Production-aware coding agents
33. Programmatic tool calling
34. Semantic review of AI-edited prose
35. Tests as data over tests as code

Caution

36. Agent instruction bloat
37. Agent skill cognitive surrender
38. AI-accelerated shadow IT
39. AI-driven documentation sprawl
40. Blanket frontier model usage
41. Coding throughput as a measure of productivity
42. Human-in-the-loop theater
43. Meat proxy
44. Naive context compression
45. Stacked pull requests

Platforms

Adopt

46. VictoriaMetrics

Trial

47. Kafka share groups
48. KServe
49. LibreChat
50. LiteLLM
51. Microsoft Copilot Retrieval API
52. Restate
53. Talos Linux
54. Temporal
55. vCluster
56. VictoriaLogs

Assess

57. Agent Client Protocol (ACP)
58. Agent Plugins
59. agentgateway
60. AWS DevOps Agent
61. Claude Managed Agents
62. Decision models
63. KAITO
64. llm-d
65. Netris
66. Omnigent
67. VAST DataStore

Caution

—

Tools

Adopt

68. mise
69. OpenAI Codex
70. OpenCode

Trial

71. Act
72. Agent Package Manager
73. ast-grep
74. Beads
75. Chaos Mesh
76. Cube Core
77. Figma Code Connect
78. FluxMarkdown
79. Git AI
80. Linear
81. mattpocock/skills
82. OpenSpec
83. OpenWiki
84. Pi
85. Sourcebot
86. Timefold Solver

Assess

87. AgentsView
88. Apple container
89. Deepsec
90. DeepSeek Harness (DSH)
91. Entire CLI
92. FreeToken
93. GLM
94. Habit Hooks
95. Herdr
96. Hunk
97. nono.sh
98. OpenResearch CLI (orx)
99. Safe Chain
100. Swamp
101. Vale
102. Varlock
103. zizmor

Caution

—

Languages and Frameworks

Adopt

104. Open Data Contract Standard (ODCS)

Trial

105. BoTorch
106. Deep Agents
107. Kagent
108. OpenFGA
109. Presidio
110. Promptfoo

Assess

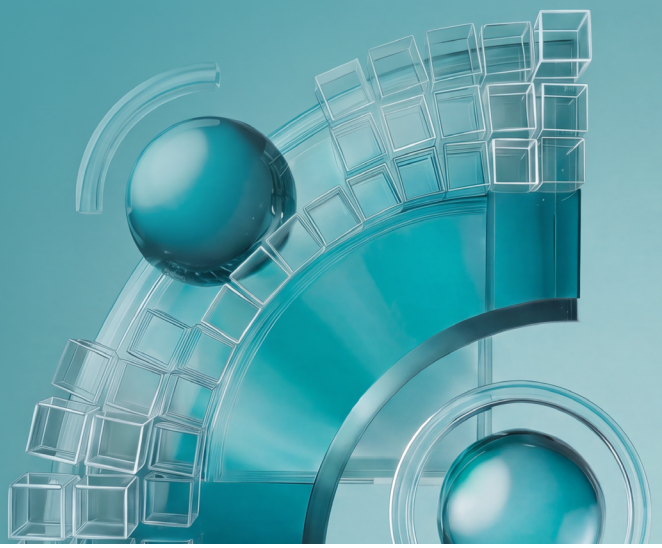
111. Apple Core AI
112. CALM
113. Hindsight
114. Quint
115. SolidJS
116. spartan/ui
117. TLA+

Caution

—

© Thoughtworks, Inc. All Rights Reserved.

Techniques



Adopt

1. Agent sandboxing
2. Mutation testing
3. Path-to-production mapping
4. Post-quantum cryptography
5. Progressive context disclosure
6. Threat modeling

Trial

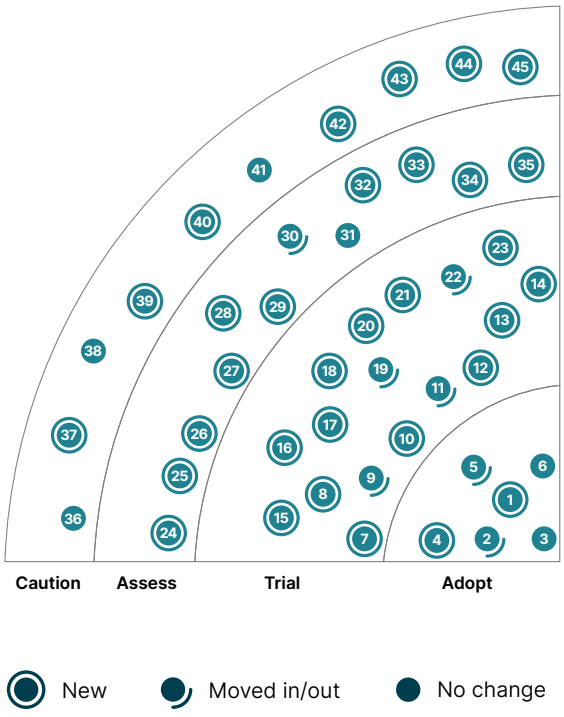
7. Agent plugin marketplace
8. Agent-maintained knowledge bases
9. Anchoring coding agents to a reference application
10. Coding harness for general-purpose agents
11. Feedback flywheel
12. Graph-grounded codebase reasoning
13. Grilling
14. Interactive visual plans with agents
15. Managed agents runtime
16. Open-weight models
17. Risk-tiered human oversight
18. Self-improving harness
19. Small language models
20. Stateful guardrails
21. Subagents for better context management
22. Toxic flow analysis for AI
23. Unattended overnight agentic builds

Assess

24. Agent pipeline responsibility
25. Agent suggestion box
26. Context profiling
27. Enterprise harness
28. Foundation models for structured data
29. Harness quality evaluation
30. Local coding models
31. MITRE ATLAS
32. Production-aware coding agents
33. Programmatic tool calling
34. Semantic review of AI-edited prose
35. Tests as data over tests as code

Caution

36. Agent instruction bloat
37. Agent skill cognitive surrender
38. AI-accelerated shadow IT
39. AI-driven documentation sprawl
40. Blanket frontier model usage
41. Coding throughput as a measure of productivity
42. Human-in-the-loop theater
43. Meat proxy
44. Naive context compression
45. Stacked pull requests



1. Agent sandboxing

Adopt

We've previously recommended sandboxed execution for coding agents. As AI agents become increasingly autonomous and capable of acting not only in coding environments but also in business systems, restricting their operational scope has become critical. We now consider agent sandboxing a sensible default for all agents. Running agents in isolated environments with restricted privileges and access to external systems reduces the risk and limits the impact of unexpected behavior, unintended actions or malicious exploitation. We recommend teams apply a risk-based approach to sandboxing, informed by threat modeling of their agents and systems. Although there is no one-size-fits-all solution, recent security headlines reinforce the importance of applying established security practices, such as the principle of least privilege, when building and running agents.

2. Mutation testing

Adopt

Mutation testing evaluates test suite quality by automatically introducing small, artificial bugs into application code and verifying that at least one test fails. If tests pass despite the code change, the mutation survives, revealing gaps in test coverage or assertion quality. As teams adopt AI coding assistants, the volume of AI-generated code and tests increases rapidly, making traditional code coverage metrics easier to inflate with superficial tests that lack meaningful validation. This technique acts as a guardrail, helping ensure generated tests don't simply pass without catching logic changes.

Beyond dedicated frameworks, AI agents can now perform mutation testing autonomously during execution by directly modifying code, running tests and evaluating assertions in their loop. Because automated mutation runs are time-consuming, developers should target specific internal packages or rely on incremental checks to run tests only on recently changed code.

3. Path-to-production mapping

Adopt

Path-to-production mapping is a standard practice at Thoughtworks that has become newly relevant with the advent of agentic coding. Increased code throughput can expose or amplify bottlenecks elsewhere in the path-to-production, making it essential for teams to optimize the end-to-end flow of software rather than coding speed alone. The activity is most often done in a workshop that includes everyone involved in designing, developing, releasing and operating the software. First, participants list the steps in the process on a whiteboard, from the developer workstation all the way to production. In a second pass, they capture pain points. The activity is often eye-opening, as participants identify delays, risks and inconsistencies they weren't previously aware of. Tackling those bottlenecks is key to improving overall delivery productivity.

4. Post-quantum cryptography

Adopt

Post-quantum cryptography (PQC) refers to cryptographic algorithms designed to protect digital data against attacks by future quantum computers. As quantum computing advances, the threat to conventional encryption becomes increasingly real: data encrypted today can be harvested now and decrypted later, making key exchange the most urgent area to migrate. With NIST's standards now finalized (ML-KEM, ML-DSA and SLH-DSA), governments worldwide target migration between 2030 and 2035.

© Thoughtworks, Inc. All Rights Reserved.

Modern infrastructure, languages (for example, Java since JDK 24) and frameworks are increasingly providing native PQC support in their latest versions. Enabling it, though, requires upgrading from older releases or adjusting configurations. Organizations should mandate quantum-resilient cryptography for greenfield projects while accelerating migration roadmaps for legacy systems.

5. Progressive context disclosure

Adopt

Progressive context disclosure is a context engineering technique for providing information to LLMs on demand. The system bootstraps the LLM with essential context and allows the LLM's responses or user input to trigger the loading of additional data. We're moving progressive context disclosure to Adopt because the practice is nearly universal in serious applications, although it takes many forms.

In coding, one example is to keep your AGENTS.md lean and provide links to documentation that the agent can look up when needed. Coding harnesses also use this approach to defer the full context cost of MCP tools and agent skills until they're used. Business applications might use vector RAG or GraphRAG to retrieve additional information based on user input. In each case, progressive context disclosure can reduce token consumption and costs and help counter context rot.

6. Threat modeling

Adopt

Threat modeling is an exercise for identifying potential threats to your systems. It's a core security practice for all software development, especially for generative AI applications, whose distinctive security risks, such as prompt injection and autonomous action, may be missed by rule-based checks and scans. Threat modeling should be a regular, human-led part of the software development lifecycle, with support from agents. The discussion helps surface undocumented behaviors and build stakeholder understanding.

We recommend starting with known threats and mapping them to your context. Useful resources include the MITRE ATLAS catalog of AI threats, MAESTRO as a companion to frameworks such as STRIDE and toxic flow analysis for reasoning about where prompt injection might enter. A session should end with explicit decisions about where nondeterministic model behavior is acceptable, where it isn't and where deterministic guardrails are needed to limit risk and harm.

7. Agent plugin marketplace

Trial

An agent plugin marketplace is a means of distributing harness assets across an organization. The need arises when multiple teams adopt coding agents and want to share skills, instructions, MCP servers, subagent definitions and other assets. Copying these between repos provides no easy way to publish, discover or update them. We've implemented an agent plugin marketplace using documented Git repositories, with plugins in the Agent Plugins format that coding agents consume directly or that are managed as dependencies with Agent Package Manager. Anthropic popularized the pattern with its Claude Code plugin marketplace, and hosted registries such as Tessl's and community collections like wshobson/agents have adopted the idea. We see agent plugin marketplaces as a practical way to make your enterprise harness available across the organization.

© Thoughtworks, Inc. All Rights Reserved.

8. Agent-maintained knowledge bases

Trial

With agent-maintained knowledge bases, a team keeps its shared knowledge in linked plain-text files, often in an Obsidian vault, and uses a coding agent to maintain them. Andrej Karpathy's LLM Wiki describes the pattern: raw sources the agent reads but never changes, a wiki it maintains and a schema that guides how it does it. OpenWiki applies a narrower version of the idea to a single code repository's documentation. The technique is practical today because general-purpose coding agents can already read and edit files effectively, while Git provides version control. Many people start with a personal "second brain." We recommend taking the next step and sharing one across a team, although expanding it into an organization-wide knowledge base takes more than pointing a coding agent at a folder of files. Generated knowledge can drift from reality, and without regular human effort to keep it accurate, the knowledge base risks creating AI-driven documentation sprawl.

Several of our teams use variations of this approach, with product managers, designers, developers and their agents all querying the same knowledge base. These teams find their information more connected and accessible.

9. Anchoring coding agents to a reference application

Trial

Anchoring coding agents to a reference application addresses the challenge of application template decay by separating lean, structural scaffolding templates from evolving platform configuration rules. Previously, we focused on using live, compilable reference applications to keep generated code in sync with current practices. We're now seeing teams successfully extend this pattern to AI-driven scaffolding.

Traditional code generators and scaffolding CLI tools quickly fall behind as platform guidelines, dependencies and architectural best practices change. Instead of hardcoding these rules into static templates, coding agents query centralized, up-to-date information sources, such as an OpenWiki or structured agent-maintained knowledge bases, at runtime to retrieve current conventions and settings. Consolidating evolving platform rules into unified, agent-accessible repositories helps agents generate code that follows current standards without requiring continuous updates to the underlying scaffolding tools.

10. Coding harness for general-purpose agents

Trial

Teams building agents for non-coding work, such as document processing, operations support or migration planning, have typically started with an agent framework and assembled the agent loop themselves. Increasingly, we see teams instead build on a coding harness for general-purpose agents, reusing the runtime of a mature coding agent. Popular coding agents now expose their runtime programmatically through the Claude Agent SDK, the Codex SDK and app-server, OpenCode's server and SDK or Pi's SDK and RPC mode. These harnesses already handle complex aspects of the agent loop, including compacting context on long-running tasks, persisting sessions and running headlessly with the same agent skills, subagents and instructions used in interactive coding-agent sessions. Compared with agent frameworks, the trade-off is less control in exchange for a runtime that's already been tested extensively. We've used this approach to build an internal accelerator for .NET application migrations.

© Thoughtworks, Inc. All Rights Reserved.

However, these harnesses differ in their capabilities. Codex and the Claude Agent SDK provide sandboxing and approval policies, while Pi deliberately leaves guardrails to extensions, making it easier to adapt but requiring more work. Because these runtimes are designed for software development, their system prompts and built-in tools tend to steer agents toward writing and running code. Teams should review these defaults for their intended domain. They should also monitor costs, since coding-agent loops can consume substantial numbers of tokens, and large built-in prompts add to every turn. That's one reason minimal harnesses such as Pi are attractive.

11. Feedback flywheel

Trial

In the previous Radar, we described the feedback flywheel as a way to improve a coding agent's harness from earlier sessions. Most developers update AGENTS.md files and agent skills when an agent gets something wrong, but what one developer learns may not be shared with the rest of the team. The feedback flywheel adds a question at the end of each session, asking whether anything should change in a shared artifact, along with a retrospective item where the team decides what to update. Feedback sensors show which checks keep failing, an agent suggestion box collects problems agents report and tools such as Backpass propose edits based on past sessions for a human to accept or reject. We're moving feedback flywheel to Trial because harness edits are now routine.

The flywheel needs to remove as well as add. Rules that only accumulate become agent instruction bloat, and a sensor that produces only noise should go. We recommend continuously checking which artifacts are still used and up to date. Context profiling shows whether edits are increasing what each session loads. Where a self-improving harness has an agent run the loop, the feedback flywheel makes it a team habit, with agents proposing changes and people choosing which changes to make.

12. Graph-grounded codebase reasoning

Trial

As context-heavy workflows become more common, sending full source files or unstructured text to LLMs increases token costs. Graph-grounded codebase reasoning addresses this by representing code, documentation and relationships as a structured graph, allowing agents to retrieve relevant subgraphs as context. Tools such as Graphify capture structural dependencies, helping agents identify relationships that text-based retrieval may miss.

However, graph grounding comes with trade-offs. Recent research suggests agents can sometimes perform worse with graph-based context than with simpler tools such as find and grep. We've placed the technique in Trial to encourage readers to explore when graph-based context improves efficiency and accuracy, building on our previous blip on code intelligence as agentic tooling.

13. Grilling

Trial

Grilling is a technique in which a coding agent interviews you about a plan before acting to resolve potential areas of ambiguity. The best-known example comes from mattpocock/skills, but other skills libraries have their own versions. One benefit is that grilling discourages coding agents from guessing where your intent is unclear. It surfaces assumptions the coding agent might otherwise make on its own. Another advantage is that it helps counter codebase cognitive debt. The grilling process forces you to examine your own assumptions, leaving you with a better understanding of the implementation plan that can be captured in decision records. Our teams have found grilling a useful addition to their agentic coding practice.

© Thoughtworks, Inc. All Rights Reserved.

14. Interactive visual plans with agents

Trial

Interactive visual plans with agents are self-contained HTML or MDX plans with diagrams, user flows and wireframes that teams can explore before implementation. Our teams find that stepping through these artifacts makes it easier to verify intent and review complex changes than reading long text plans. Our teams warn against using this technique for "big upfront designs" and favor a more iterative approach, breaking work into smaller, verifiable steps to build understanding along the way. The same approach can provide a visual recap of completed work, but our teams find recaps most useful when the implementation deviates from the original plan. We recommend using interactive visual plans with agents to build shared understanding and establish a human review point before agents implement changes.

15. Managed agents runtime

Trial

Managed agents runtime is the practice of running software delivery agents on a platform that manages their execution, infrastructure and operational needs rather than relying on a developer's workstation. The runtime hosts the agent loop and provides sandboxed compute, triggers, credential handling, session state and observability, leaving teams to define the model, prompt, tools and skills. Long-running, asynchronous work is well-suited to background agents that pick up tasks from issues, pipelines or schedules and submit pull requests for review.

Coding tools are converging on this approach, with Cursor cloud agents, the GitHub Copilot coding agent and OpenAI Codex cloud tasks all running in remote sandboxes. Claude Managed Agents offers a general-purpose equivalent. Our internal security team runs a daily agent that triages endpoint detection false positives and proposes rule changes. "Managed" doesn't necessarily mean someone else's cloud. At one client, we run custom agents on a self-hosted runtime in a continuous loop across multiple repositories, fixing code smells flagged by scanners and upgrading dependencies along with any required code changes.

A hosted service is often the easiest place to start, but organizations with data residency, security or integration constraints may need their own platform. Either way, we recommend treating the runtime as production infrastructure, with agent sandboxing and version-controlled agent definitions.

16. Open-weight models

Trial

Open-weight models, where model weights, architectures and inference code are publicly released, have reached a significant inflection point. Attracted by their lower total cost of ownership, we've been evaluating the latest models from families such as DeepSeek, GLM and Kimi for production delivery work requiring complex reasoning and high-quality code generation, and we're seeing good results.

Beyond cost efficiency, open-weight models provide deployment flexibility, freedom from vendor lock-in and stronger data sovereignty. Engineering teams can right-size their deployments, from lightweight flash models to hundred-billion- or trillion-parameter mixture of experts (MoE) models, and perform private fine-tuning using domain data. Although self-hosting requires greater operational effort, major cloud providers now offer managed open-weight models that lower the barrier to adoption. Rather than treating them merely as budget options, organizations should evaluate open-weight models for a broad range of enterprise workloads, well beyond coding.

© Thoughtworks, Inc. All Rights Reserved.

17. Risk-tiered human oversight

Trial

Risk-tiered human oversight aligns review depth with what a change could break. Reviewing everything with the same intensity ends in human-in-the-loop theater. We suggest systematically triaging changes. The riskiest or hardest-to-reverse changes are reserved for humans to apply. Intermediate-risk changes go through human review. Low-risk changes can be auto-merged by coding agents through a continuous integration pipeline.

Two strategies can help us decide which changes need more review. Zones in the codebase require elevated review with particular files or directories, while LLM-based change classification analyzes the impact of changes but is fallible. Meta's published process combines these two approaches. Our teams find risk-tiered human oversight an effective way to make the best use of scarce human attention.

18. Self-improving harness

Trial

A self-improving harness uses agents to analyze execution traces and evaluation feedback, then propose or implement improvements to the system surrounding the model, including prompts, context strategies, workflows, skills, tools, memory and middleware. The idea isn't new: techniques such as the feedback flywheel use successes and failures from coding-agent sessions to improve the harness but still rely on humans to interpret the feedback, decide what to change and update the system. Recent research explores a more self-directed approach. Meta-Harness and VERSE use execution traces and evaluation results to improve harness code; VERSE also evolves the optimizer's own tools and procedures. Lilian Weng's overview traces this progression from prompt optimization to harness evolution.

Observability and programmability are essential to this transition. Observability helps agents diagnose failures from execution traces and connect changes to measurable outcomes, while programmability gives agents access to components they can inspect, modify and test, expanding the scope for self-improvement. Pi's extensions, Claude Code Mods and DeepSeek Harness (DSH) illustrate this direction through extensible context, workflows, tools and harness components.

Greater openness creates more opportunities for improvement, but gains still require validation. We recommend bounded experiments with regression testing, while humans define objectives, set modification boundaries and assess whether improvements generalize to new tasks.

19. Small language models

Trial

We continue to see a shift in the industry away from a "bigger is better" mindset toward improving artificial intelligence through high-quality training data, model distillation and quantization. Small language models (SLMs) have become increasingly useful for specific tasks that don't require the computational resources of larger models. As organizations face rising costs from running LLMs, teams are recognizing the value of choosing the right model for the task.

Compact models are also viable for edge computing, as Apple Core AI demonstrates by running small foundation models directly on devices. Our teams have found SLMs effective for narrow, high-frequency tasks that don't require extensive knowledge or complex reasoning, such as summarization,

© Thoughtworks, Inc. All Rights Reserved.

structured data extraction, intent classification for request routing and basic coding. Unless an agentic use case requires multi-step reasoning or reliable tool calling, teams should consider an SLM as a lower-cost, lower latency alternative.

20. Stateful guardrails

Trial

Stateful guardrails extend LLM guardrails from single messages to whole conversations. Instead of checking only the current user message or the final model response, they evaluate a compact transcript of recent turns against policy-specific checks and maintain strike or ban state across the session. This matters because unsafe intent can emerge gradually: each message may look acceptable on its own, while the conversation as a whole reveals attempts at policy evasion, jailbreaking or prompt extraction. In practice, stateful checks run as a runtime gate before the agent acts, working alongside conventional single-turn input and output guardrails.

Evaluating conversation history on every turn adds latency and cost, and strike or ban thresholds need tuning to prevent false positives from locking out legitimate users. Our teams have used this technique in a production conversational assistant for sensitive workflows, tracing guardrail decisions in Langfuse. We recommend that teams running multi-turn assistants in production try stateful guardrails rather than relying on per-message checks alone.

21. Subagents for better context management

Trial

Subagents for better context management is the practice of delegating work to subagents to keep it out of the orchestrating agent's context. Exploration, tool output, retries and intermediate reasoning stay in the subagent, and the main session receives a short summary. Most people first use subagents to run tasks in parallel or form a role-based team of coding agents. Protecting the orchestrator's context is another reason to use them. Whatever remains in the orchestrator's context competes for its attention on subsequent turns. Pulling a subagent's full transcript into the main thread to check its progress undoes this benefit. Useful guidelines include running two to four subagents at a time and keeping tasks that require the same understanding of the code within one agent, so multiple agents don't each have to rebuild that understanding.

Using subagents to protect the orchestrating agent's context is a form of context engineering that's easy to try because most coding agents support subagents, and the change is largely in the instructions. We've used cheaper models for subagents handling narrow tasks such as research, potentially reducing costs compared with blanket frontier model usage. We're placing the technique in Trial because its benefit hasn't been measured systematically. The evidence so far comes from practitioners' accounts of their own sessions, without detailed token accounting. We've also observed that the same model with the same prompt uses subagents differently depending on the harness. Teams trying this approach should use context profiling to check whether the orchestrator's context actually shrinks.

22. Toxic flow analysis for AI

Trial

As AI agent capabilities expand, teams are increasingly deploying systems that face a lethal trifecta of security risks: access to private data, exposure to untrusted content and the ability to communicate externally. This combination creates opportunities for attacks such as prompt injection and tool poisoning. To address these risks, we've moved toxic flow analysis for AI to Trial.

© Thoughtworks, Inc. All Rights Reserved.

While general threat modeling provides a broad security foundation, toxic flow analysis focuses on identifying unsafe data paths through agentic systems. These risks aren't limited to standard integrations. They can also arise from malicious agent skills that appear useful but contain hidden instructions to exfiltrate sensitive data. We've successfully used tools such as Agent Scan to perform this analysis, helping teams identify and mitigate unsafe data paths before they're exploited.

23. Unattended overnight agentic builds

Trial

Unattended overnight agentic builds refers to the practice of giving a coding agent a feature-sized piece of work when the team leaves for the evening and reviewing what it produced the next morning. We've done this with epic-level features and infrastructure tasks that can take hours. One agent is enough, without a coding agent swarm, although it may use subagents. One Thoughtworks team spends the day collaborating on plans and specifications and leaves the coding to agents overnight. "Overnight" is in the name because it's hard to fake. A team that can do this will likely run similar jobs during the day too. (A team that doesn't leave an agent coding unwatched overnight probably isn't running fully autonomous work during the day either.)

Like microservice prerequisites, this practice has entry requirements, and pushing teams to meet them is a big part of its value. Agents need sandboxed execution, work specified up front as in spec-driven development and feedback sensors so they can correct their own mistakes. A night's work can be too large a batch to review effectively line by line, so teams need a harness they've tested to validate the changes instead of relying on human code review. Complementary techniques include running the agent in a managed agent runtime and letting agents keep the build green by fixing pipeline failures themselves. We've placed this in Trial for teams that have those foundations; for other teams, it's a goal that justifies building them.

24. Agent pipeline responsibility

Assess

Agent pipeline responsibility incorporates coding agents into the team's accountability for keeping the build green. An agent that can't see the build status can't respect a team agreement not to push on red, while a human relaying failure logs into a chat window becomes a meat proxy.

We recommend making the build state part of the agent's context. Expose pipeline status and logs through the agent's tools so it can see what went wrong and help fix it. Enforce important rules mechanically with pre-push and server-side gates to prevent agents from pushing broken changes. The increased commit rate from coding agents is leading teams to rethink assumptions around build discipline through techniques such as risk-tiered human oversight. Include the role of agents in your build policy when conducting path-to-production mapping.

25. Agent suggestion box

Assess

An agent suggestion box gives a running agent somewhere to report the problems it encounters, such as a bug in a tool or a dependency missing from its environment. When an agent's only output is a pull request or a structured verdict, it may work around the problem without anyone hearing about it. The next session then hits the same problem again. The box can be a Slack channel, a memory store or an issue tracker. Lovable calls its version a "vent tool," and agents-feedback offers one for coding agents. Memory can help an agent avoid a broken tool, but a suggestion box gets the tool fixed. The technique works for business agents in production as well as for coding agents.

© Thoughtworks, Inc. All Rights Reserved.

We've used an agent suggestion box with an agent that starts a fresh session each day to draft detection engineering rules. A second agent deduplicates the reports. The same report from several sessions is a clear signal, although some reports are subjective or transient. With the box in place, runs went more smoothly, and we had a list of improvements to make. The suggestions still need to be reviewed and acted on, for example, through self-improving harness. We're placing agent suggestion box in Assess because our experience comes from one agent. Lovable merges about ten fixes a day from its vents but reports that roughly half the fixes its triage agent proposes are false positives.

26. Context profiling

Assess

As token costs have risen and the industry has left the tokenmaxxing fad behind, developers are looking for ways to manage token usage. The old advice that you should always measure before optimizing still applies. Context profiling means analyzing where your token spend is going (for example, *are tokens being spent on file reads or MCP tool calls?*). Intuition is a poor guide to identifying which optimizations are worthwhile, especially when you factor in caching, model tiers and promotional pricing. Mature tooling hasn't yet emerged, but you can make do by analyzing your agent sessions or data from tools such as Entire CLI and AgentsView.

In our experience, the biggest savings often come from simple changes, such as clearing sessions at the right time, tuning cache parameters or quietening chatty build scripts. Yet we often see developers reaching for complex solutions such as compression proxies without measuring their impact.

27. Enterprise harness

Assess

An enterprise harness is the organization-wide layer of what coding agents consume managed collectively. It includes skills, tools, connectors to internal context, approved technologies, architecture decision records, path-to-production guidance, hooks and security policies. We observe that as multiple teams in an organization adopt coding agents, they need common context to complement each team's curated shared instructions. It's like a developer experience platform, but for agent experience. We've seen organizations go as far as building a custom coding agent on top of OpenCode, but we recommend sharing assets through open standards and distributing them via an agent plugin marketplace. There's no consensus yet on how to do this well, so keep the enterprise harness thin and federated, leave room for experimentation and harvest what works.

28. Foundation models for structured data

Assess

Enterprise AI has focused heavily on text and vision, while much business data remains locked in relational databases. Foundation models adapt transformers to structured data, pretraining on an organization's event sequences and transaction logs to learn reusable representations of business behavior.

Unlike traditional gradient-boosted models that require extensive feature engineering and retraining, these models can support multiple downstream tasks with lightweight fine-tuning. Tools such as TabPFN and TabFM can improve risk assessment, predictive analytics and customer retention directly from tabular or relational data. Given their potential to unlock business insights without complex feature pipelines, we suggest assessing this technique.

© Thoughtworks, Inc. All Rights Reserved.

29. Harness quality evaluation

Assess

Nowadays, organizations and teams come up with custom harnesses tailored to their projects, including agent skills, plugins, AGENTS.md and deterministic guardrails. Harness quality evaluation has become an important technique. Guardrails and harnesses help a coding agent course-correct and improve overall quality, but only if they actually work.

A harness usually includes both deterministic and non-deterministic components. For non-deterministic harnesses, our teams use PluginEval to evaluate coding agent plugins or skills. For deterministic harnesses, we introduce a test suite to enable TDD-style development of any guardrails, ensuring each one actually works and providing evidence of its effectiveness. We recommend running evaluations on a scheduled basis (such as nightly or alongside CI).

30. Local coding models

Assess

In previous editions of the Radar, we put local coding assistants in Caution because the available models were too limited for complex software engineering contexts and weren't a compelling solution to the concerns they were intended to address. With coding agents now being adopted much more widely than 18 months ago and local open-weight models maturing rapidly, our teams have found local coding models effective in addressing blanket frontier model usage.

Models that can run locally, such as Qwen3.8-27B, have proved particularly useful for workloads that don't require a frontier model, including codebase indexing, structured log parsing and summarization, initial coding tasks and automated reviews of Git diffs and commits. Running these tasks locally can avoid incremental cloud token costs while also enabling long-running background processing and providing an offline fallback.

31. MITRE ATLAS

Assess

Agentic systems and coding tools introduce new architectures and emergent security threats. MITRE ATLAS is a knowledge base of adversarial tactics and techniques targeting AI and ML systems. More focused than the broader MITRE ATT&CK framework and designed to complement it, ATLAS provides a taxonomy of threats for ML pipelines, LLM applications and agentic systems.

We've found that without a shared vocabulary, security risks are often overlooked or reduced to a checkbox exercise. This is where ATLAS can help. Grounded in research on real incidents and technology patterns, teams can use the framework to support threat modeling. Teams may also find it a natural complement to control frameworks such as SAIF, helping describe the evolving threat landscape for AI systems.

32. Production-aware coding agents

Assess

We've long said that being conscious of production helps developers write better code, and we're finding the same applies to coding agents. We recommend production-aware coding agents, which means giving coding agents appropriate access to production observability data and tools. This context helps them plan effective changes, review the impact of deployed changes and assist with investigating production issues. A growing tooling category is making this access practical. This recommendation is similar to agent pipeline responsibility, extending the feedback sensors that keep

© Thoughtworks, Inc. All Rights Reserved.

agents self-correcting during a coding session beyond pre-commit checks and into production. We believe successful development has multiple layers of feedback, and successful agentic engineering gives agents access to all of them.

33. Programmatic tool calling

Assess

Programmatic tool calling is a tool-use pattern where the model writes code instead of issuing one tool call at a time. Rather than making a round-trip through the model for each operation, it combines tool calls into a program that runs in an execution environment, looping, filtering and shaping results. Only the selected results or the final output are returned to the model's context.

The concept appears in several implementations, although the naming is still unsettled. Cloudflare and Pi call it Code Mode; Anthropic calls it programmatic tool calling; and an earlier Anthropic engineering article describes code execution with MCP. DeepSeek Harness (DSH) also provides a closely related PTC Mode.

Implementations differ mainly in where the execution environment runs: in a model provider's container, the agent harness or, more recently, through a Code Mode MCP server. The latter allows MCP clients to use programmatic tool calling without implementing their own execution runtime. However, it can reduce the granularity of client-side approvals, so authorization and approval for individual operations must still be enforced by the server and its tool handlers.

34. Semantic review of AI-edited prose

Assess

Coding agents produce and revise prose artifacts such as specifications, ADRs and plans far faster than humans can review them. Text diffs bury intent under rewording and formatting noise, making it hard to see whether a plan has changed significantly. Semantic review of AI-edited prose adds a layer on top of the text diff: an LLM analyzes changes in meaning, such as altered facts, conflicting statements, redefined acceptance criteria and changes in scope. Much like automated PR review but for prose, it can be implemented as an agent skill that runs whenever a coding agent edits a document. We've seen Thoughtworks teams use this technique to reduce cognitive load and review fatigue, helping reviewers better understand changes to planned work before implementation.

35. Tests as data over tests as code

Assess

Tests as data over tests as code means expressing test cases as structured data, typically YAML or JSON, rather than as test code. A project-specific test harness reads that data and executes the underlying code: the harness provides the building blocks, while the test cases define how they're assembled. The technique isn't new; BDD with Gherkin is a prime example. What makes it relevant again is that AI coding agents now write more code, including tests. We've found that a defined schema better constrains what they produce, making tests easier to generate and review, with less AI-generated test code.

A few of our teams have also seen a compelling use case in legacy modernization, where the same test cases expressed as data can run against both legacy and modern code by swapping the harness adapter. This helps teams verify that the modernized system behaves like the original, at least for the cases covered by the tests. Using the technique with AI is still relatively new to us, and we haven't applied it broadly. However, the results so far are promising, and we think it's worth assessing.

© Thoughtworks, Inc. All Rights Reserved.

36. Agent instruction bloat

Caution

Instructions for coding agents, such as AGENTS.md guidance, process skills and workflows, tend to grow longer over time. Although each new rule might be useful in isolation, this accumulation often leads to agent instruction bloat. The instructions become long, contradictory and stale, especially if teams engage in agent skill cognitive surrender and import external instruction sets wholesale. First, convert any instructions you can into deterministic hooks or checks. Then, reevaluate instructions against each new model and delete those it no longer needs; progression testing, rather than regression testing. Anthropic removed over 80% of Claude Code's system prompt for its latest models with no measurable loss, as frontier models now follow instructions more reliably. Progressive context disclosure keeps the remaining instructions out of the way until they're needed.

37. Agent skill cognitive surrender

Caution

There is a growing trend of agent skill cognitive surrender, an antipattern in which teams adopt large, community-curated skill libraries or heavyweight AI agent frameworks without fully understanding what they do. Much like blindly importing large software dependencies, developers can treat these skills as ready-made capabilities and lose sight of how the underlying system works. This isn't a new problem; similar examples include copy-pasting code without understanding it or over-relying on generic design patterns. However, the proliferation of AI tools has made the issue much more apparent.

Beyond the loss of system understanding, surrendering critical evaluation introduces security risks: unreviewed prompt instructions can make their way into execution pipelines. Skills from different sources may also conflict when they weren't designed to work together. It's important to inspect and review skill instructions before adapting them rather than using general-purpose agent skills indiscriminately. As underlying models become more capable, generic community skills can become unnecessary or obsolete, making it more valuable to curate domain-specific skills tailored to a project's context.

38. AI-accelerated shadow IT

Caution

We observe many noncoders graduating from no-code platforms to coding agents such as Claude Code and OpenAI Codex. We celebrate people solving their own problems, but AI-accelerated shadow IT, whether built by noncoders or developers, comes with risks. First among these is security. Second, software originating from outside traditional development usually lacks testing, deployment pipelines and observability. This particularly matters if one person's productivity tool is adopted by their team or department. Organizations should prioritize governance with limits and guidelines on AI-generated applications. In our experience, it's better to provide colleagues with environments and enterprise harnesses for safe experimentation than to extinguish their enthusiasm. Be firm on security, and give useful tools a clear path to support.

39. AI-driven documentation sprawl

Caution

AI-driven documentation sprawl occurs when teams use AI agents and spec-driven development tools such as Spec-kit or BMAD to generate large volumes of Markdown files and documentation. Because AI can produce polished, professional-looking docs in seconds, the result can create an illusion of

© Thoughtworks, Inc. All Rights Reserved.

software quality and rapid progress. Unlike source code, however, documentation rarely passes through automated quality gates or CI tests. As projects evolve, repositories rapidly accumulate verbose, unmaintained and contradictory specs that developers lack the time to clean up, resulting in a chaotic "big ball of documentation mud."

This overabundance of obsolete documentation is particularly problematic in agentic workflows. When coding agents read through a repository littered with outdated architectural decision records or conflicting specification files, these redundant and contradictory documents consume valuable context and provide unreliable information. Rather than aiding development, obsolete context can lead AI tools astray and result in poor code generation. To maintain agent effectiveness, teams must treat Markdown as a first-class artifact, spending deliberate human effort to prune, refine and consolidate generated context files so the agent's context remains crisp, relevant and authoritative.

40. Blanket frontier model usage

Caution

We've sometimes caught ourselves defaulting every task that requires an LLM to the most capable model available, paying no attention to cost, latency or required capability. We call this blanket frontier model usage. Frontier models can cost significantly more and run more slowly than their more nimble counterparts; routine work such as exploration, simple edits or summarizing is often better served by a smaller model or even open-weight models. You can switch tiers manually, per subagent or through harness settings that switch automatically based on task complexity or stage. Each model has its own prompt cache, so you can also switch at natural boundaries, like a new session, a subagent or the handoff from planning to implementation, rather than per request. A warning on our warning, though: more capable models sometimes finish tasks using fewer tokens, so use context profiling to check whether your optimization is working.

41. Coding throughput as a measure of productivity

Caution

In a previous Radar, we warned against using coding throughput as a measure of productivity. As the industry pushes for AI adoption and seeks tangible metrics, we increasingly see activity-based metrics such as the percentage of code generated by AI or tokens consumed treated as badges of honor. Although such metrics may indicate adoption levels, they're poor proxies for real productivity gains. To measure meaningful productivity, we continue to recommend established frameworks such as DORA metrics and collaboration quality, along with tying improvements directly to business outcomes. Treating activity metrics as targets rather than signals risks falling prey to Goodhart's law. We've seen developers generating AI busywork simply to hit these targets.

42. Human-in-the-loop theater

Caution

Human-in-the-loop theater is keeping a human in an agent's loop as an empty approval step rather than a real check. The human approves actions, prompts or diffs they can't realistically evaluate.

The approval step functions as an accountability sink: no one truly owns the decision. Our advice is instead to triage. Reserve human review for changes a human can genuinely judge, using risk-tiered human oversight. For the rest, shift from reviewing individual changes to steering the system with mutation testing, property-based testing, formal models such as TLA+ and Quint and agent-assisted semantic review.

© Thoughtworks, Inc. All Rights Reserved.

43. Meat proxy

Caution

A meat proxy is a person who unthinkingly passes on unverified LLM output to a colleague, forwarding it without reading, validating or adding substantive judgment. It's a violation of good AI etiquette and, in our unfortunate experience, a growing problem. When you act as a meat proxy, you shift the verification burden downstream onto the other person and shirk your own responsibility for assuring quality work. This is similar to human-in-the-loop theater, except that it's fake authorship instead of fake approval. In neither case is the human adding value. KDE's draft AI guidelines put it well: "Don't be a meat proxy."

44. Naive context compression

Caution

As token costs rise, teams increasingly turn to tools like Headroom, RTK and Tamp to shrink their prompts. However, we advise against naive context compression, which can introduce several subtle problems. First, it often interferes with prompt caching; reducing token counts while missing cache hits can lead to higher actual bills. Second, aggressive compression causes information loss, degrading the performance and accuracy of agentic loops. Third, some tools rely on an LLM to summarize context during compression, adding latency and costs that defeat the purpose.

Before adopting these tools, profile your context usage and rigorously evaluate both net cost and task accuracy; token reduction means nothing if prompt caches break or agent performance degrades. Opt for tools that compress tool outputs while preserving references to the raw data and that prune history only in one shot at task boundaries, to protect cache prefixes.

45. Stacked pull requests

Caution

Stacked pull requests break a large change into an ordered chain of small, dependent pull requests (PRs). Rather than targeting the main repository branch, each subsequent PR targets the branch directly below it in the stack. We've seen a surge in interest in this technique recently due to the rapid adoption of AI-assisted coding tools. Teams adopt this approach to reduce review latency and developer idle time. By evaluating small, isolated layers, reviewers face lower cognitive overhead. Tooling has also improved to support this workflow; GitHub recently introduced native support.

Despite better tooling, we've placed this technique in the Caution ring because it's often a workaround for poor upfront decomposition. Stacking doesn't create decomposition problems; it exposes ones that were already there. The risk isn't the technique itself; it's using it as a substitute for that planning. Habitual, undifferentiated stacking can easily become an excuse to write one enormous change. We maintain that the better habit is to specify, split, size and deliver work as small, independent units.

© Thoughtworks, Inc. All Rights Reserved.

Platforms



Adopt

46. VictoriaMetrics

Trial

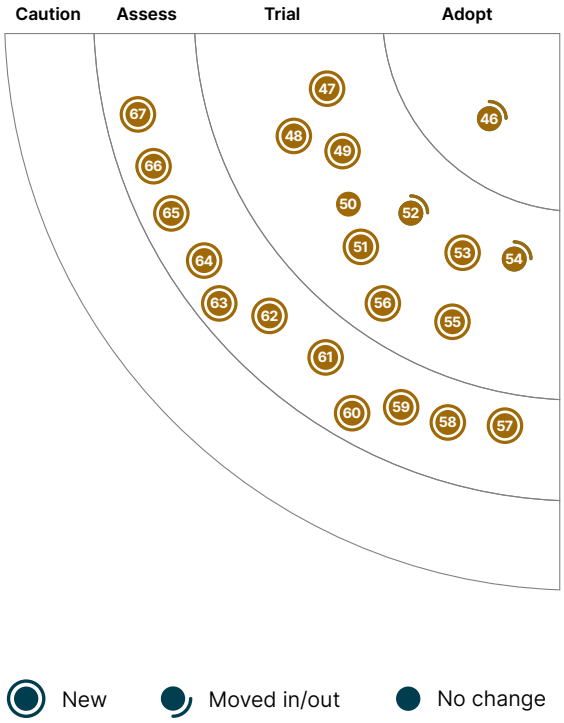
47. Kafka share groups
48. KServe
49. LibreChat
50. LiteLLM
51. Microsoft Copilot Retrieval API
52. Restate
53. Talos Linux
54. Temporal
55. vCluster
56. VictoriaLogs

Assess

57. Agent Client Protocol (ACP)
58. Agent Plugins
59. agentgateway
60. AWS DevOps Agent
61. Claude Managed Agents
62. Decision models
63. KAITO
64. llm-d
65. Netris
66. Omnigent
67. VAST DataStore

Caution

—



New Moved in/out No change

46. VictoriaMetrics

Adopt

We've moved VictoriaMetrics to Adopt because it has matured into a reliable choice for organizations operating large-scale, self-managed observability platforms. As a high-performance, open-source time-series database, it handles high-volume and high-cardinality metrics workloads that can challenge Prometheus. In one of our projects, a VictoriaMetrics cluster handles five million samples per second with 20 to 30 labels per metric, demonstrating its suitability for large-scale deployments.

The platform offers a straightforward single-node version for smaller workloads as well as a clustered architecture that separates ingestion, querying and storage into independently scalable components. It integrates with existing observability architectures by supporting native OpenTelemetry ingestion, Prometheus APIs and remote write support, allowing teams to adopt it without replacing their existing instrumentation and dashboards. Whether used as long-term storage behind Prometheus or as a complete metrics stack with components such as vmagent and vmalert, VictoriaMetrics is a compelling option for infrastructure where retention, cardinality or cost has become a challenge.

47. Kafka share groups

Trial

Kafka share groups (introduced in KIP-932, production-ready since Kafka v4.2) bring a new cooperative event consumption model to Kafka, enabling queue-like behavior. Historically, Kafka's traditional consumer groups have imposed a concurrency bottleneck because the number of active consumers can't exceed the number of partitions. Share groups remove this constraint: individual records are handed to any available consumer in the pool under a per-record acquisition lock, which the consumer must explicitly acknowledge. On failure or timeout, the record is released and redelivered to any consumer in the group, up to a configurable maximum number of delivery attempts. This provides at-least-once delivery with the possibility of duplicate-processing. Records that exhaust their delivery attempts are archived; dead-letter queue support (KIP-1191) is not yet available.

The tradeoff is that share groups don't guarantee per-partition ordering because records are dispatched and redelivered independently. However, for teams that previously had to run both Kafka and an external queue system, share groups can simplify the architecture by consolidating workloads onto Kafka.

48. KServe

Trial

KServe is an open-source Kubernetes operator that manages predictive and generative AI inference through custom resources. Rather than acting as an inference engine itself, it orchestrates serving runtimes such as vLLM, SGLang and NVIDIA Triton, separating the lifecycle of a model service from the engine that executes it. As organizations build AI factories, operating inference across Kubernetes GPU infrastructure becomes a fleet-management problem rather than a matter of running individual model servers. KServe addresses this with two custom resources: InferenceService for conventional machine learning and simpler LLM workloads, with optional Knative-based scale-to-zero, and LLMInferenceService for advanced LLM serving. Built on llm-d and the Gateway API Inference Extension, LLMInferenceService supports multi-node inference, prefill/decode disaggregation and KV-cache-aware routing that directs requests to replicas already holding the relevant cached prefix.

© Thoughtworks, Inc. All Rights Reserved.

KServe is a CNCF incubating project, and several of our clients use it in production. Our teams prefer it over other tools in this space when building in-house inference platforms on Kubernetes. LLMInferenceService is newer than InferenceService, and its ecosystem is still evolving.

49. LibreChat

Trial

LibreChat is an open-source, self-hosted AI chat platform that lets people across an organization access models from multiple providers through a single interface. Like Open WebUI, it offers single sign-on, role-based permissions and RAG over uploaded documents. Its agent features make it more than a chat frontend. Administrators decide which MCP servers are available. Users whose role allows it can then build agents on those servers without writing code and share them through a built-in marketplace.

One of our teams built a client's internal AI platform on LibreChat. The team wrote MCP servers for Jira, Confluence, Salesforce, Grafana and the client's own order, inventory and revenue systems. The client's staff then built the agents themselves: sales created one to check availability and book orders, while operations built one to manage contracts. That experience is why we're placing it in Trial. It gives people outside engineering a controlled way to access vetted AI tooling, which is what we've recommended as the answer to AI-accelerated shadow IT. ClickHouse acquired LibreChat in November 2025, and the project remains MIT licensed.

50. LiteLLM

Trial

LiteLLM has become a widely used open-source AI gateway. Several of our teams route their model calls through it, either as a standalone proxy or as a sidecar. It supports more than 100 LLM providers and offers routing, provider translation, authentication, budgets and guardrails. Its proxy core is being rewritten from Python to Rust to improve performance while retaining the same Python SDK and configuration.

We're moving LiteLLM from Languages and Frameworks to Platforms in this edition because we now see it more as infrastructure than a library. We're keeping it in Trial rather than Adopt largely because of security concerns: at the time of writing, more than 25 CVEs have been reported this year alone. Teams using it should apply security patches promptly. To also govern MCP and agent-to-agent traffic, consider combining it with agentgateway.

51. Microsoft Copilot Retrieval API

Trial

Microsoft Copilot Retrieval API is a streamlined solution for retrieving data from the Microsoft ecosystem for retrieval-augmented generation (RAG). The API retrieves relevant text extracts from SharePoint, OneDrive and content exposed through Copilot connectors. It applies security trimming based on the calling user and respects access controls defined within the tenant. Developers can build organization-aware AI agents without maintaining custom data stores and retrieval pipelines, leaving tasks such as chunking, indexing, data synchronization, reindexing and access control to Microsoft. The Microsoft Copilot Retrieval API is particularly useful for AI systems that need organizational knowledge already stored in Microsoft tools, and we've had a positive experience adopting it.

© Thoughtworks, Inc. All Rights Reserved.

52. Restate

Trial

Restate is a durable execution platform for building stateful, fault-tolerant applications. Because it journals every step of execution, a service that fails resumes where it left off. Restate also guarantees exactly-once communication between services. An application combines Basic Services for stateless functions, Virtual Objects for concurrent stateful entities and Workflows for multi-step processes.

When we last wrote about Restate, we were assessing it on a large insurance system. It's now a core part of that system, automating an entire insurance business process. The team relies on its state recovery and concurrency control and hasn't had to write its own recovery code. That experience is why we're moving it to Trial. Restate suits long-running processes that wait on a human in the loop or retry many times against legacy systems. It fills the same role as Temporal, which also moves to Trial in this edition. The SDKs are MIT licensed, but the server uses the Business Source License, which rules out offering it to third parties as a managed service.

53. Talos Linux

Trial

Talos Linux is a minimal, immutable Linux distribution for running Kubernetes. It has no SSH, package manager or interactive shell. You configure nodes declaratively with talosctl through an mTLS-secured gRPC API, and upgrades replace the entire OS image. This design reduces the attack surface and configuration drift associated with general-purpose distributions such as Ubuntu and makes each node a replaceable appliance, in the spirit of immutable servers.

One of our teams runs self-managed Kubernetes on Talos across a large bare-metal fleet for an AI cloud platform and found the control plane easy to provision. Most organizations use a managed service such as Amazon EKS and never interact with the node OS, so Talos is primarily for teams that operate their own nodes. Bottlerocket takes a similar approach on AWS, and Flatcar Container Linux also has traction. In 2026, Yardi acquired Sidero Labs, the developer of Talos. Talos remains open source under MPL-2.0, but teams should watch where the new owner takes it.

54. Temporal

Trial

Temporal is a durable execution platform for complex business processes that require strong reliability guarantees. Temporal provides workflows with retries, intermediate state persistence, configurable timeouts, human-in-the-loop checkpoints, resumability and visibility into execution progress. It offers both an open-source self-hosted option and a managed cloud service.

We've found compelling use cases for Temporal when building internal developer platforms and control planes for AI cloud infrastructure. These systems typically involve multiple long-running infrastructure provisioning tasks whose state must be reconciled later. Temporal's built-in retries, flexible worker queue scheduling, resumability, visibility and durable execution help teams build these types of systems more resiliently.

55. vCluster

Trial

vCluster provides virtualized Kubernetes control planes for multi-tenant environments. It offers a middle ground between namespace-level multi-tenancy, where tenants share cluster-wide APIs and custom resource definitions (CRDs), and provisioning completely independent clusters, which

© Thoughtworks, Inc. All Rights Reserved.

increases operational overhead. With vCluster, each tenant gets its own Kubernetes API server, controller manager, data store, RBAC and CRDs. The control plane runs as a lightweight workload on the underlying infrastructure, allowing standard tools such as kubectl, Helm and GitOps controllers to interact with each tenant independently.

We've successfully used vCluster to build managed Kubernetes services and AI cloud environments where balancing tenant isolation against operational costs is critical. It's particularly attractive for AI platforms where different customers may need their own GPU operators, schedulers, inference frameworks and CRDs. For trusted workloads, virtualized control planes can share underlying compute resources. For customer-facing GPU environments, dedicated worker nodes can provide stronger workload isolation while retaining consolidated control plane management. Given its success in production client environments, we're placing vCluster in Trial.

56. VictoriaLogs

Trial

Developed by the creators of VictoriaMetrics, VictoriaLogs is an open-source log management platform optimized for high-throughput ingestion, compact storage and fast querying of large log volumes. Rather than requiring predefined indexes, its storage engine organizes and compresses log fields into blocks. It uses Bloom filters and timestamp indexes to avoid unnecessary scans and parallelizes queries across CPU cores.

We've had positive experiences using VictoriaLogs while building observability for an AI cloud infrastructure system. At this scale, Kubernetes services generate large volumes of telemetry, making ingestion throughput, query performance and storage efficiency critical considerations. Compared with traditional search-based logging platforms like Elasticsearch, VictoriaLogs offers a capable logging backend with less operational complexity and lower resource overhead. We've also found it particularly attractive for high-cardinality structured logs and arbitrary field searches when compared with Grafana Loki, because it doesn't require the same degree of label design. VictoriaLogs also includes the LogsQL query language for full-text search, transformations and aggregation. It integrates with Grafana for interactive exploration and alerting, while compatibility with OpenTelemetry, Elasticsearch and Loki ingestion protocols allows teams to adopt it without tying applications to a proprietary logging pipeline.

57. Agent Client Protocol (ACP)

Assess

While CLI-based coding agents excel at repo-wide implementation and workflow execution, developers still need inline access to agents within their IDEs for rapid code exploration, interactive bug fixing and context-aware editing. Traditionally, this requires individual IDE plugins, creating integration overhead and locking developers into specific editor-agent pairings. Agent Client Protocol (ACP) addresses this by establishing an open, standard boundary between AI agents and host environments, whether that's VS Code, JetBrains, a terminal window or a CI/CD pipeline.

Much like the Language Server Protocol standardized language support across editors, ACP relies on JSON-RPC to define agent lifecycle and interaction primitives, including session creation and resumption, streaming updates, permission requests and cancellation. ACP complements existing interoperability layers: while the Model Context Protocol (MCP) handles agent-to-tool integration, ACP handles client-to-agent control, making the complete agent harness a swappable component.

© Thoughtworks, Inc. All Rights Reserved.

58. Agent Plugins

Assess

Agent Plugins is an open standard for packaging reusable agent components into portable plugins. The first version, launched by a cross-vendor collaboration including Amazon, Microsoft and OpenAI, defines a shared format for agent skills and Model Context Protocol (MCP) servers. Although the specification is deliberately minimal, an open standard is an important step forward for an ecosystem that has so far been fragmented. We've found that the "small interoperability floor" makes it easy to extend an existing skills repository so that it doubles as an Agent Plugin.

59. agentgateway

Assess

agentgateway is a high-performance gateway for LLM-related traffic, hosted by the Linux Foundation. It supports LLM inference, MCP and A2A communication through a common protocol in both standalone and Kubernetes versions. Core features include cost- and latency-based routing, fallbacks, spending limits, guardrails, MCP tool control and OpenTelemetry observability.

We're particularly interested in the Kubernetes version because of its deep integration with self-hosted LLM infrastructure. It can use Kubernetes metrics to route requests to a GPU-backed replica that already has the relevant context cached in memory or to a less utilized server based on actual GPU usage. These capabilities make agentgateway worth evaluating for organizations running self-hosted inference stacks.

60. AWS DevOps Agent

Assess

AWS DevOps Agent is a managed AWS agent platform for AI-driven incident investigation. When an alarm, ticket or chat message triggers an investigation, the platform maps application resources and their relationships, correlates telemetry with recent deployments and code changes, tests hypotheses and presents likely root causes and mitigations for an engineer to act on. Built-in integrations with Datadog, Dynatrace, New Relic, Splunk and Grafana let alerts from those tools trigger investigations and allow the platform to query their telemetry through MCP-based integrations. Connections to GitHub, GitLab, ServiceNow, PagerDuty and Slack, along with custom MCP servers, extend its reach to on-premises and proprietary systems.

Our teams are experimenting with it in production to investigate live alarms. It's early, but the results look promising, suggesting the potential for faster root-cause analysis and less operational toil. Our main reservation is cost transparency. The platform charges per agent-second spent investigating an incident, making costs difficult to predict. Spending can add up quickly without close monitoring. The service has only been generally available since March 2026, so we recommend assessing it with human validation of every finding, tight control over which events trigger investigations and active cost tracking.

61. Claude Managed Agents

Assess

Claude Managed Agents is Anthropic's hosted service for running general-purpose agents. Teams define each agent's model, prompt, tools, MCP servers and skills. Anthropic provides the rest of a managed agents runtime: the agent loop, sandbox, scheduling and observability. Agents

© Thoughtworks, Inc. All Rights Reserved.

run on a schedule or when an application starts a session through the API. Scheduled runs are suited to recurring background work, such as triaging alerts and proposing fixes as pull requests for human review.

Teams can get started without writing code through the console or use code-based integrations for greater flexibility and control. Agent, environment and schedule definitions can be stored in files applied through a CI pipeline. A credential vault injects credentials into outbound requests without exposing them to the agent. One of our internal teams chose Claude Managed Agents over other agent platforms, including Gemini Enterprise Agent Platform, because it was easier to understand and faster to deliver with.

We're placing it in Assess because our experience is limited to that team and the service is in beta. Unlike Amazon Bedrock AgentCore, it supports only Anthropic's agent harness and Claude models. Anthropic charges for session runtime as well as tokens. Because the service stores sessions, it isn't eligible for Zero Data Retention. Self-hosted sandboxes can keep tool execution within your network, but orchestration and tool results still pass through Anthropic.

62. Decision models

Assess

A new class of AI models called decision models has emerged since the last Radar. TypeSafe AI released Jev, which it calls a "system one" model, built to make fast, structured decisions. Many other models have followed, including Laya, Nimble and Kev. In contrast to deliberative reasoning models, decision models output predefined structured values instead of generated text. Rather than using reinforcement learning from human feedback (RLHF) to optimize for human preferences, Jev uses reinforcement learning from calibrated decisions (RLCD) to optimize for well-calibrated decisions, where predicted probabilities reflect how often an outcome actually occurs. Their smaller, structured outputs enable faster and cheaper processing.

Decision models receive a state and a set of questions. Every question asked against the state must map to one of three types: Noul for yes/no questions
that return the probability of a "yes" answer, Choice for categorized multiple-choice options or Score for evaluations on an ordered list of levels or a numerical scale. Decision model use cases include Jev-as-a-judge as an alternative to LLM-as-a-judge, model and tool routing, policy gates and bulk labeling. We recommend teams avoid overenthusiastic LLM use and define evaluation criteria to determine whether a decision model can replace an LLM for classification use cases.

63. KAITO

Assess

KAITO (Kubernetes AI Toolchain Operator) is a Kubernetes operator that automates the infrastructure needed to self-host open-weight models on your own clusters. Instead of manually configuring GPU nodes and model servers, teams declare a Workspace, KAITO's custom resource for a single model deployment, specifying the model and GPU instance type. The KAITO controller then provisions matching GPU nodes through Karpenter NodePools, estimates GPU memory requirements, tunes the runtime and serves the model through vLLM, exposing an OpenAI-compatible endpoint via a Kubernetes Service.

Our teams began exploring KAITO on AKS to host small language models at lower cost and latency. Previously, running Ollama and vLLM directly on GPU VMs had left them managing drivers, scaling and node configuration manually. Managing the model lifecycle through the same declarative manifests as other workloads promises to reduce much of that toil.

© Thoughtworks, Inc. All Rights Reserved.

KAITO's scope is deliberately narrow, though. It handles node provisioning and model deployment rather than the broader inference stack, where KServe is more mature and offers capabilities such as KV-cache-aware routing. KAITO is still a CNCF Sandbox project. Its GPU node auto-provisioning is most mature on AKS, and it supports only Ampere-generation or newer NVIDIA GPUs. We think it's worth assessing, particularly for teams that want to self-host models using their existing Kubernetes tooling.

64. llm-d

Assess

llm-d is an open-source distributed inference serving stack designed to operate model servers such as vLLM and SGLang across multiple Kubernetes nodes. Traditional web applications often rely on stateless routing and horizontal scaling, but LLM inference is highly stateful due to key value (KV) caching. Rather than treating model-serving pods as interchangeable HTTP backends, llm-d uses inference-aware routing to coordinate requests across multiple accelerators, extending optimization beyond individual model servers.

Built around Kubernetes and the Gateway API Inference Extension, llm-d offers several capabilities tailored to stateful AI traffic. It supports model-aware and KV-cache-aware routing, using factors such as cache affinity, queue depth and predicted latency to guide scheduling decisions. A notable feature is prefill and decode disaggregation, which allows operators to place compute-intensive prefill workloads on one set of GPUs and memory-bandwidth-bound decoding tasks on another. This can improve hardware utilization and reduce inference costs. Teams building large-scale GPU inference platforms should assess llm-d as an emerging foundation for their inference layer, particularly when looking to scale while meeting strict latency objectives.

65. Netris

Assess

Netris is an out-of-band network automation and multi-tenancy platform for physical data centers and AI infrastructure. Using a declarative, infrastructure-as-code model accessible through a UI, an API, Kubernetes CRDs and Terraform, it simplifies network management by abstracting switch configurations into cloud-style virtual private clouds (VPCs), virtual networks, access control lists (ACLs) and Layer 4 load balancers. A central controller manages policies while lightweight agents execute changes on switches and data processing units (DPUs). The controller operates out-of-band as an automation control plane, so packet forwarding can continue uninterrupted even if the controller goes offline.

We're highlighting Netris for its support of high-bandwidth AI workloads. It automates routed Ethernet VPN (EVPN) fabrics, manages flow control for RDMA over Converged Ethernet (RoCE) and orchestrates NVIDIA BlueField DPUs for hardware-enforced multi-tenancy. It also integrates with NVIDIA UFM for InfiniBand partition keys (PKeys) and NMX-C for NVLink domains, helping manage multi-tenant boundaries across different AI fabrics through a unified management layer. We're placing Netris in Assess for teams building automated private clouds or AI data centers, although teams should evaluate compatibility with their switch operating systems, such as SONiC, Cumulus Linux and Arista EOS, as well as the boundaries of bare-metal orchestration support.

© Thoughtworks, Inc. All Rights Reserved.

66. Omnigent

Assess

Omnigent is an open-source meta-harness, released by Databricks, that sits above coding agents such as Claude Code, OpenAI Codex and Pi, as well as agents defined in YAML. It provides a common layer for capabilities that each harness otherwise implements differently. Policies can cap spending or require approval before a risky action, while an OS sandbox restricts file system and network access and swaps credentials into outgoing requests, so the agent can use them without reading them. Sessions persist, allowing a run to resume where it stopped. Omnigent can also communicate with agents through the Agent Client Protocol (ACP) instead of wrapping each CLI.

On a laptop, Omnigent gives a developer an inbox of parallel sessions, much as Herdr does. As a shared server, it adds single sign-on, live session sharing and administrator-set policies, with sessions running locally or in cloud sandboxes such as E2B. This suits organizations that want consistent controls regardless of which coding agents their developers choose. We aren't yet sure what the right abstraction over coding agents is, and the project still describes itself as alpha. Teams that want consistent policies across several coding agents should evaluate Omnigent with both caveats in mind.

67. VAST DataStore

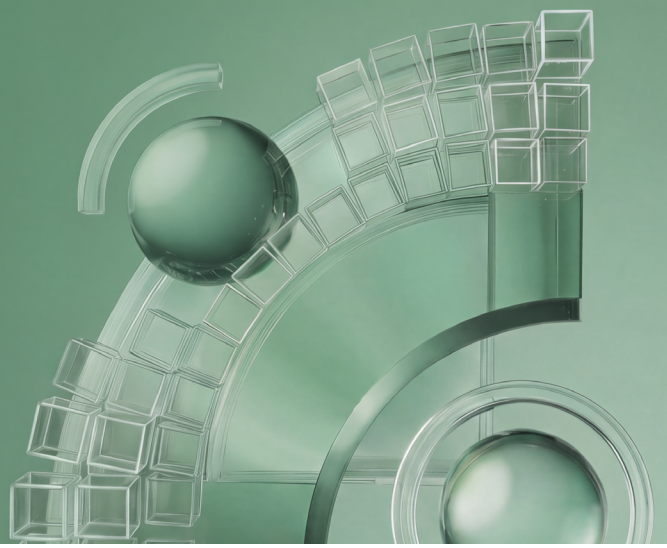
Assess

VAST DataStore is the storage layer of the VAST Data platform. It provides file (NFS and SMB), object (S3) and block (NVMe/TCP) access through a unified namespace. Its disaggregated shared-everything architecture separates stateless compute nodes from shared NVMe storage, allowing capacity and performance to scale independently. We're assessing it because storage throughput increasingly affects how efficiently expensive GPU clusters operate during training, checkpointing and model loading. DataStore lets workloads retain conventional NFS/POSIX or S3 access while using NVMe and RDMA for faster data transfer, offering an alternative to specialized parallel file systems. VAST is also working with NVIDIA on S3 over RDMA and shared KV-cache storage for disaggregated inference.

Compared with Ceph, which offers open-source block, file and object storage on commodity hardware, DataStore is a proprietary platform designed for NVMe, RDMA and AI workloads. However, it's expensive and ties you to VAST's broader data platform, so it may be more than teams need unless they're training models or running large-scale inference. We think VAST DataStore is worth assessing for teams designing large GPU clusters for AI and high-performance computing (HPC) workloads, where data loading, checkpointing and metadata-heavy access to billions of files can leave expensive GPUs waiting on storage.

© Thoughtworks, Inc. All Rights Reserved.

Tools



Adopt

68. mise
69. OpenAI Codex
70. OpenCode

Trial

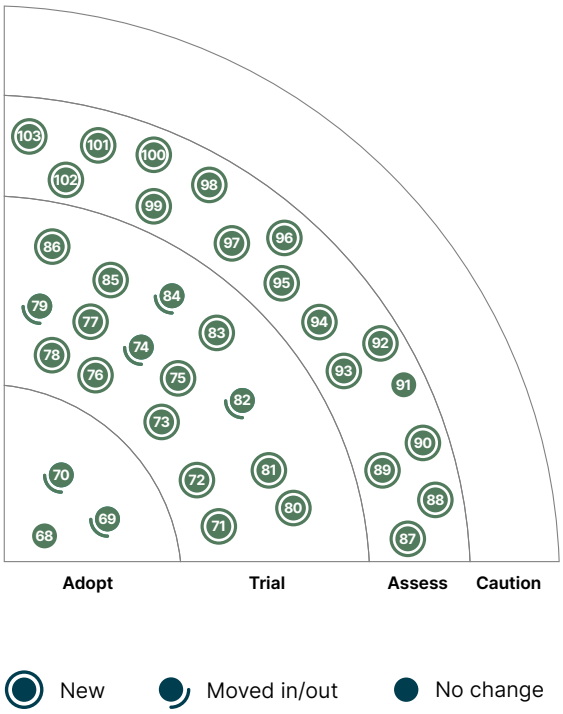
71. Act
72. Agent Package Manager
73. ast-grep
74. Beads
75. Chaos Mesh
76. Cube Core
77. Figma Code Connect
78. FluxMarkdown
79. Git AI
80. Linear
81. mattpocock/skills
82. OpenSpec
83. OpenWiki
84. Pi
85. Sourcebot
86. Timefold Solver

Assess

87. AgentsView
88. Apple container
89. Deepsec
90. DeepSeek Harness (DSH)
91. Entire CLI
92. FreeToken
93. GLM
94. Habit Hooks
95. Herdr
96. Hunk
97. nono.sh
98. OpenResearch CLI (orx)
99. Safe Chain
100. Swamp
101. Vale
102. Varlock
103. zizmor

Caution

—



68. mise

Adopt

Since our last assessment, mise has evolved from a high-performance alternative to asdf into a default frontend for the development environment. We're keeping it in Adopt because it consolidates three fragmented concerns (tool and language versioning, environment variable management and task execution) into a single high-performance, Rust-based tool, configured through a declarative mise.toml file. mise is easy to set up and works well with CI/CD pipelines. It also adds a layer of supply chain security through integration with Cosign and GitHub artifact attestations, which is often missing from other version managers.

For teams looking to standardize their developer environment setup, mise has become our recommended default. In polyglot environments with multiple microservices, this is especially useful when codebases adopt new language versions at the same time. Best of all, mise also works with existing language-specific tooling, so teams do not need to migrate all at once.

69. OpenAI Codex

Adopt

OpenAI Codex moves to Adopt, joining Claude Code and OpenCode as CLI-first coding agents our teams use day to day. Its desktop app is popular with non-coders. Codex has the features you'd expect for enterprise adoption: OS-level sandboxing, tiered approvals, central configuration and compliance management. The underlying agent loop is Apache-2.0 licensed and can be used for the technique we call coding harness for general-purpose agents. Although open source, Codex is corporate-run rather than community-governed. It employs the classic challenger's gambit of compatibility: it makes it easy to import configuration, skills and sessions from other agents and to use models besides OpenAI's GPTs. That makes Codex a solid choice for running open-weight models.

70. OpenCode

Adopt

Since our last Radar, OpenCode remains one of the most prominent open-source coding agent harnesses. We've seen its adoption grow significantly alongside the rise of capable open-weight models. Teams appreciate its customizability and headless/server mode, which make it an attractive foundation for organizations building an enterprise harness. On the downside, its session-sharing and collaboration capabilities still lag behind competing harnesses. OpenCode has become a popular choice for teams pairing coding agents with self-hosted, open-weight models and local coding models, but we still recommend evaluating emerging alternatives such as Pi.

71. Act

Trial

Act runs GitHub Actions workflows locally, replacing the commit–push–wait loop with faster feedback. We mentioned it in our GitHub Actions blip back in 2021. Coding agents give Act new relevance because the growing volume of commits and pushes can cause build contention. Act lets coding agents run a workflow locally before pushing, bringing CI directly into their generative loop and supporting agent pipeline responsibility.

© Thoughtworks, Inc. All Rights Reserved.

Our teams find that Act handles most GitHub Actions workflow development, although it isn't a perfect replica of the GitHub Actions environment. For example, its default images lack some runner tools, and it doesn't support concurrency, permissions and environment secrets. Paired with static checking such as zizmor, Act helps coding agents keep the build working when they change the pipeline itself.

72. Agent Package Manager

Trial

Agent Package Manager is an open-source dependency manager for AI agent configuration. It works via a manifest that declares plugins, skills, instructions and MCP servers. We use it to publish versioned skills from a central repository instead of copy-pasting files around. It treats agent context as a supply chain, installing from Git repos or an agent plugin marketplace and scanning for hidden Unicode that could hijack behavior. Its lock files record content hashes and provenance, while its SBOMs export to CycloneDX and SPDX formats. Auditing detects drift, and a policy file constrains sources, scopes and primitives, making an enterprise harness reproducible and governed. It's still a pre-1.0, community-driven project, albeit one sponsored by a major player, so we place it in Trial rather than Adopt and expect competing solutions to follow.

73. ast-grep

Trial

ast-grep is a command-line tool for structural code search, linting and rewriting. Patterns are written as ordinary code with wildcards for any syntax node and match against the syntax tree rather than text, using tree-sitter parsers for more than 20 languages. Linting and rewriting rules are written in a declarative YAML format. One of our teams used these rules as part of the harness around a coding agent, checking the agent's code against a set of custom rules and feeding the results back to the agent.

ast-grep is similar to the tools behind code intelligence as agentic tooling, but it doesn't build an index or require a language server. Because it skips indexing, it's a low-friction first step for a large codebase before committing to heavier tools. Unlike most tools in that group, it also has uses that don't involve agents: developers can run the same searches, lint rules and rewrites themselves. ast-grep covers much the same ground as Comby, which we featured in 2021. Most IDEs already offer AST-based search, but they don't replace the custom lint rules and rewrites that set ast-grep apart.

74. Beads

Trial

Beads is a Git-backed issue tracker that coding agents can share to coordinate their work. In place of ad hoc Markdown plans, it gives agents a persistent, structured task graph. Because Beads stores tasks and their dependencies alongside the code, agents retain their plans and newly discovered tasks across sessions and context compaction. Each agent can claim the next available task without colliding with others.

We're moving Beads to Trial because many of our developers now use it by default with coding agents. It complements team issue trackers such as Linear, which teams use to plan, prioritize and track work. Beads holds the finer-grained tasks agents create as they work, and it syncs both ways with Linear, Jira and GitHub Issues.

© Thoughtworks, Inc. All Rights Reserved.

Since we last featured it, Beads has reached 1.0 and moved to the Gas Town Hall organization. It's maintained by Gas City, the company behind Gas Town. Dolt is now its only storage backend, and additions such as an HTTP API, work leases and synchronization between stores make Beads heavier than the lightweight tracker it started as. Upgrades migrate the schema in place, and clients running different versions can't share a store, so teams should pin a single version for each shared store and review the instructions Beads writes into AGENTS.md.

75. Chaos Mesh

Trial

Chaos Mesh is an open-source Chaos Engineering toolkit for injecting controlled failures into cloud-native systems. Chaos Mesh defines fault simulations as declarative Kubernetes custom resources, allowing teams to use familiar tooling and manage chaos experiments through version control. We've found that its range of simulated failure scenarios, reproducibility and blast radius controls make it suitable for repeatable, automated testing at scale. This includes environments with GPU-dense AI infrastructure, where validating resilience can be particularly challenging but important.

76. Cube Core

Trial

As analytics data is increasingly consumed by both custom applications and AI agents, defining metrics directly in dashboards or individual SQL queries often leads to duplication and metric drift. Cube Core is an open-source semantic layer that addresses this challenge by allowing teams to define metrics, dimensions, joins and access rules as code. By creating a governed, reusable semantic model, teams can define business metrics once and expose them consistently across BI tools, embedded analytics and AI workflows through standard APIs.

We've used Cube Core successfully to implement a semantic layer in recent projects. The tool has an active codebase and contributor community. If you're struggling with inconsistent data definitions across multiple consumption layers, we recommend evaluating Cube Core to standardize analytics access.

77. Figma Code Connect

Trial

Generic design-to-code tools often turn frames into raw markup that ignores a team's existing components, resulting in code that needs to be rewritten. Figma Code Connect addresses this disconnect by mapping Figma components directly to production UI components in code. Instead of guessing component APIs, AI tools and developers can use these mappings to generate code with your actual component signatures and props. This structured approach grounds code generation in the existing design system.

This tool is particularly relevant as teams move from prompt-based code generation to more controlled, agentic workflows. By pairing Figma Code Connect with the Figma MCP server and coding agents, developers can provide agents with context from the actual codebase. This helps shift free-form code generation toward more deterministic, production-compliant component assembly, reducing code review iterations and the risk of architectural drift.

© Thoughtworks, Inc. All Rights Reserved.

In our experience with mobile design systems, we've used this workflow to map components across iOS and Android, auto-publishing to Figma after main-branch releases and using continuous integration (CI) to detect drift. However, Figma-to-code generation is only as reliable as the underlying design system. Generated output still requires guardrails, such as lint checks to block hardcoded colors and snapshot baselines on every pull request. Maintaining mappings for every variant also requires ongoing effort, and the tool currently operates at the component level rather than validating complete product screens. Layout logic, state and navigation therefore still require human oversight.

78. FluxMarkdown

Trial

FluxMarkdown is a macOS QuickLook plugin that renders Markdown files directly in the Finder preview panel. While other options like QLMarkdown exist, FluxMarkdown offers comprehensive support for syntax highlighting, mathematical notations via KaTeX and complex diagrams using Mermaid, Graphviz and Vega.

We've found FluxMarkdown useful for inspecting documentation because it eliminates the need to open a dedicated editor just to view a rendered diagram or equation. As developers increasingly rely on command-line interfaces and autonomous agents in their workflows, viewing rendered Markdown files can still require opening an IDE. FluxMarkdown provides a lightweight alternative that lets developers preview these files without leaving Finder.

79. Git AI

Trial

Git AI is an open-source Git extension designed to track AI-generated code in repositories. It links AI-written lines to the agents, models and prompts that generated them. Using checkpoints and hooks, the tool tracks incremental code changes throughout development. Each checkpoint contains a diff between the current state and the previous one, marking lines as either human- or AI-authored. This approach provides more detailed attribution of AI contributions than simply counting lines of code at the time they're added.

The extension uses an open standard based on git notes to track AI-generated code. As AI-generated code becomes more common in everyday development, maintaining accountability and understanding the original intent behind code becomes increasingly important. Both developers and AI agents can query archived sessions to retrieve the original context and design decisions. Given its potential value for accountability and maintainability in agentic workflows, we've moved Git AI to Trial.

80. Linear

Trial

Linear is an issue tracker that pairs the easy setup of Trello with the planning features teams usually turn to Jira for. Many startups have used it in place of Jira for years, and its recent appeal comes from how well it works with coding agents. Agents join a Linear workspace as members, so teams can assign them issues, mention them in comments and see the reasoning behind each change. Cursor, OpenAI Codex and Devin can all take on work this way, while Linear's remote MCP server lets tools such as Claude Code read and update issues. Linear holds the work a team plans and prioritizes. Beads sits a level below it, tracking the smaller tasks agents create as they go. We often use Linear with clients whose teams work heavily with coding agents. Teams that choose their own tooling should consider trialing it if they want agents to pick up work directly from the backlog.

© Thoughtworks, Inc. All Rights Reserved.

81. mattpocock/skills

Trial

mattpocock/skills ("Skills For Real Engineers") is a collection of agent skills for agentic software engineering. The skills provide a suggested workflow but are explicitly designed to be small, easy to adapt, and composable. We've found them valuable for helping teams apply sound software engineering practices to agentic coding, as well as for AI-assisted analysis of requirements and assumptions, such as grilling. While we caution against adopting entire skill libraries without evaluating them (agent skill cognitive surrender), these lightweight skills are worth a look if you want more structure in your agentic software development workflows.

82. OpenSpec

Trial

OpenSpec is a lightweight spec-driven development framework that works with any coding agent. Since we featured it in Assess in April, several of our teams have reported success with it, enough to move it to Trial. We particularly appreciate how it works on brownfield codebases, retrofitting specs onto existing code. Teams report more predictable agent output and fewer unintended changes compared to bare prompting. OpenSpec's key differentiator over competitors like GitHub Spec Kit and Kiro is that it centers on a spec-anchored workflow. Changes are deltas against a persistent source-of-truth spec, so agents can rely on it as a current reference. The usual caution applies: spec files are Markdown that agents generate prolifically. OpenSpec's archiving of change deltas mitigates this, but maintaining the persistent spec still requires discipline.

83. OpenWiki

Trial

OpenWiki is a command-line interface that automatically writes and maintains a wiki for a codebase or personal knowledge repository. It synthesizes source materials into a linked Markdown wiki that other agents can use as context. OpenWiki is currently in our Trial ring as we evaluate its impact on active projects. Our experience has reinforced a familiar challenge: high-quality documentation is essential, but keeping it up to date is notoriously difficult. OpenWiki addresses this by using automated pipelines to update content as source material changes, with checkpointing to track when each piece of documentation was last generated.

OpenWiki also supports local models for private generation and the OKF format for structured documentation. For humans, it includes an interactive visualizer that presents the underlying wiki as an explorable node graph, helping them navigate and understand relationships across interconnected topics and codebase documents.

84. Pi

Trial

Pi is a minimal, open-source agent harness written in TypeScript. This blip covers the whole toolkit: the multi-provider LLM API (pi-ai), the agent runtime (pi-agent-core) and the terminal coding agent built on them (pi-coding-agent). Pi deliberately leaves out MCP, subagents, plan mode and permission prompts. You add what you need through extensions, agent skills and prompt templates. Pi can even write its own extensions. We're moving Pi to Trial because adoption has grown since the last Radar, mostly as a foundation for other tools. Security scanners and orchestration harnesses now run on Pi, often as an underlying dependency rather than a directly used tool, while its SDK and RPC modes let you embed it in other agent systems.

© Thoughtworks, Inc. All Rights Reserved.

As a coding agent used directly, Pi isn't the strongest choice, and OpenCode is better suited as an enterprise coding harness. Oh My Pi, a popular fork, adds language server integration, subagents and MCP to create a full coding agent, but it has diverged enough to evaluate as a separate tool. Our concern last time was Pi's dependence on one maintainer; in April 2026, its creator joined Earendil, which now owns Pi and has committed to keeping the core MIT-licensed while planning commercial add-ons.

85. Sourcebot

Trial

Just as developers need broad context across large, multi-repository enterprise codebases, software engineering agents need the same context to perform large-scale refactoring and navigate legacy ecosystems. Sourcebot is a self-hosted code search and understanding platform that acts as a context layer to serve both. It indexes thousands of repositories to provide fast, symbol-aware search and IDE-level navigation. With the Sourcebot MCP server, agents can search across an organization's entire codebase. Our teams have also found the paid-tier feature Ask Sourcebot useful for team collaboration. This AI chat tool enables non-developer roles to query the codebase using natural language, reducing their reliance on developers to answer codebase questions.

86. Timefold Solver

Trial

Timefold Solver is an open-source constraint optimization engine for Java and Kotlin and the modern successor to OptaPlanner. It solves logistical planning problems, such as vehicle routing, employee rostering, task assignment and maintenance scheduling, where the space of possibilities is far too large to search exhaustively. Although branded under the broader AI umbrella, it involves no training data or learned model, relying instead on heuristic search over constraints you define. It models absolute business rules as hard constraints that a returned plan can't violate and operational targets as soft constraints that it optimizes within. We've successfully used Timefold Solver in production in the supply chain domain to rapidly model complex business scenarios.

87. AgentsView

Assess

As AI-assisted coding becomes mainstream, AgentsView gives developers observability into coding sessions so they can analyze patterns and improve harness usage. AgentsView lets developers browse, search and perform context profiling on past AI coding sessions. It can automatically hook into popular coding harnesses, such as Claude Code, OpenAI Codex, OpenCode or Pi and retrieve historical sessions. It includes coding session analysis tools for metrics such as prompt-cache and cost efficiency, along with recommendations for improving harness usage. One thing we like about AgentsView is its local-first design, which makes adoption straightforward even for security-conscious organizations, while team synchronization and collaborative session reviews can be added later.

88. Apple container

Assess

Apple container is an open-source tool for running Linux containers natively on Apple silicon. Written in Swift, it uses Apple's Containerization package to run a lightweight VM for each container rather than a monolithic Linux VM. This architecture provides hypervisor-level isolation and dynamic memory allocation, returning RAM to macOS when a container is torn down. It offers a native, lightweight alternative to shared-VM managers such as Colima and avoids the licensing requirements associated

© Thoughtworks, Inc. All Rights Reserved.

with Docker Desktop. When paired with community bridges such as Socktainer for Docker API compatibility, Apple container is particularly effective for local Testcontainers setups that require fast cold starts and strict resource boundaries. The project also includes Container Machine for running persistent, host-integrated Linux environments.

89. Deepsec

Assess

Deepsec is an agent-powered vulnerability scanner that first scans a codebase to understand its architecture, then launches targeted AI reviews against high-risk areas to produce vulnerability reports. What stands out is its hybrid architecture: it combines deterministic matching to identify high-risk files with dedicated subagents that inspect them. By narrowing the model's focus to where it matters most, Deepsec can reduce token consumption and cost compared to similar tools.

90. DeepSeek Harness (DSH)

Assess

DeepSeek Harness (DSH) is an open-source runtime from DeepSeek AI for building customized coding and general-purpose agents. Rather than offering a monolithic assistant, DSH uses an "everything is a plugin" architecture powered by the Cordis kernel. The model adapter, tool registry, session log and agent loop are all plugins, organized into profiles: named plugin lists that you extend and override in your own YAML.

Cordis is a meta-framework built around spatiotemporal composability: plugins contribute registrations as reversible effects, so reloading and teardown unwind them predictably rather than leaving dangling state. The combination of this plugin architecture and spatiotemporal composability could provide a foundation for domain-specific enterprise harnesses and future self-improving harnesses.

DSH also treats observability as a first-class concern. Every model-visible interaction, including prompts, context injections, tool calls and subagent collaborations, is captured in an append-only session log. This allows runs to be inspected, resumed, forked, searched and replayed, providing a record of what a recomposed system actually did.

91. Entire CLI

Assess

Entire CLI hooks into Git to capture coding agent sessions, including prompts, transcripts, tool calls and files touched, and stores them in the repository alongside the commits they produced but outside the branch history. It works with Claude Code, Codex, Cursor, OpenCode, Pi and others. Since we last featured Entire CLI, one of our internal teams that works heavily with coding agents has used it to share every member's sessions. The team found it useful for seeing how teammates prompt, what their agents do with those prompts and how prompting affects the quality of the output. It could also help experienced developers show others how they work with agents.

Openness comes with a cost, however. Entire CLI keeps a permanent record of how each person prompts, and anyone with access to the repository can read it. When we tried introducing it to other teams, some developers pushed back. A coding agent can feel like a personal assistant, and they weren't comfortable sharing everything they asked it with the team. For those teams, AgentsView offers a local alternative, letting developers review their own sessions without sharing them. We suggest trying Entire CLI in highly collaborative teams with a strong culture of trust. It stays in Assess because our experience comes from one internal team, and we don't yet know how other teams can adopt it comfortably.

© Thoughtworks, Inc. All Rights Reserved.

92. FreeToken

Assess

FreeToken is an open-source inference engine designed to run large open-weight mixture-of-experts (MoE) models on consumer hardware. As these models become more capable, their full weights can exceed the memory limits of consumer GPUs. Unlike general-purpose local model runners, FreeToken specializes in oversized MoE models by adaptively combining GPU memory, CPU execution, host memory and PCIe bandwidth. It dynamically caches active experts on the GPU and swaps them based on which experts are activated. By coordinating this process, FreeToken enables developers to run much larger models on existing or older hardware. We're starting to see MoE offloading techniques incorporated into broader tools such as llama.cpp, which is used by Ollama, but FreeToken offers a specialized approach worth assessing.

93. GLM

Assess

As the capability gap between open and closed AI models shrinks, the GLM model family by Z.ai stands out for its performance on software engineering workloads. Using a roughly 700-billion-parameter mixture-of-experts (MoE) architecture, models such as GLM-5.3 achieve strong benchmark results in coding, reasoning and agentic tasks, driven largely by specialized post-training rather than pretraining scale alone. This makes GLM-5.3 one of the best open-weight models currently available for coding and software engineering.

We like GLM because it provides teams with an open-weight alternative that competes with proprietary frontier models and complex reasoning tasks. However, in our experience, the model can sometimes "think" too much, overanalyzing straightforward prompts. We advise teams to configure the reasoning effort based on the complexity of the workload.

94. Habit Hooks

Assess

Habit Hooks is an open-source code-quality coaching tool for AI coding agents. It runs existing static-analysis and linting tools such as Ruff, ESLint, Knip, PMD, deptry and jscpd. Instead of returning raw linter errors, it translates tool-specific rule violations into actionable forward guidance and code smell context. A bare linter violation tells an AI agent that a check has failed but might encourage a superficial change that merely satisfies the rule. Habit Hooks maps the finding to the associated code smell and deterministically injects explicit guidance into the model's prompt. This context helps the agent address the actual design problem and self-correct immediately without relying on an LLM to generate the recommendation. The tool provides a practical implementation of feedback sensors for coding agents, integrating deterministic checks into the implementation loop so agents can respond to failures before human review.

95. Herdr

Assess

Herdr is an open-source terminal multiplexer and runtime designed to manage multiple coding agent sessions. It provides familiar multiplexer capabilities such as panes, keyboard navigation, detaching and reattaching, making it easier for developers to migrate from tools such as tmux. Herdr keeps agents running in persistent terminal sessions while exposing agent-aware states such as working, blocked, done and idle. Integrations via a CLI allow agents to create panes, exchange prompts and coordinate work.

© Thoughtworks, Inc. All Rights Reserved.

As developers increasingly run parallel coding agents across multiple workspaces, keeping track of which sessions require attention and how to resume interrupted work has become a new workflow challenge. Tools such as T3 Code, Orca and Paseo provide alternative experiences for developers who prefer a GUI. Teams managing parallel agents should assess Herdr if they want persistent sessions and visible agent status in their terminal workflow.

96. Hunk

Assess

Hunk is an open-source terminal diff viewer for reviewing changes made by coding agents. It displays a GitHub pull request or a Git, Jujutsu or Sapling changeset as a multi-file review with a file sidebar and split or unified layouts. A watch mode refreshes the view as an agent edits the working tree. An agent can load Hunk's agent skill to drive a live review session, jumping to a file or Hunk and adding notes alongside the lines it changed. Reviewers can then read the agent's reasoning next to the relevant code. Those notes distinguish Hunk from syntax-aware diff tools such as Difftastic.

We aren't sure line-by-line human review of agent output will remain the norm, but many teams still need it today: skimping on it can lead to complacency with AI-generated code. The project started in March 2026 and hasn't yet reached version 1.0. Teams that review agent changes in the terminal should try Hunk and assess whether the agent's notes make reviews faster or simply give reviewers more to read.

97. nono.sh

Assess

nono.sh is an open-source security shell and lightweight sandbox designed to run AI agents without the overhead of traditional virtual machines or Docker containers. It addresses a common gap: although sandboxing is widely recommended, setup overhead can discourage teams from implementing it. nono.sh isolates agent processes directly at the OS level using native kernel features such as Landlock on Linux and Seatbelt on macOS. It operates in-process as native host processes without background daemons, blocking access to sensitive host assets, such as SSH keys, by default.

We think nono.sh is worth assessing because it lowers the barrier to process isolation. Its minimal design makes it straightforward to configure, test and integrate into existing workflows across macOS, Linux and Windows through WSL2. Security profiles can be tweaked, forked and shared across teams or the broader community through the nono registry, providing a low-overhead layer of defense for developers adopting autonomous tools such as Claude Code or OpenAI Codex.

98. OpenResearch CLI (orx)

Assess

OpenResearch CLI (orx) is an open-source harness that turns coding agents such as Claude Code into research agents that run experiments in parallel. Each experiment is isolated in its own Git worktree, so attempts don't interfere with one another. All experiments use the same run command, which keeps results comparable, and every run is pinned to an immutable snapshot of the code for reproducibility.

Because orx runs on compute you choose (local, Slurm, Kubernetes or Modal) teams can implement a method from a paper, reproduce its experiments and then test whether its results hold in their enterprise context. This can help identify the gap between academic results and enterprise practice. We think orx is worth assessing as a promising domain-specific harness for narrowing that gap.

© Thoughtworks, Inc. All Rights Reserved.

99. Safe Chain

Assess

Safe Chain is a free tool from Aikido that wraps the package managers for Node.js and Python, such as npm, pnpm, Bun, pip and uv. It sends package downloads through a local proxy that blocks versions published in the last 48 hours and any package Aikido's open threat feed lists as malicious. Worms such as Shai-Hulud have shown how quickly a compromised package can spread, and a short delay gives scanners time to catch a bad release. Most package managers now have a minimum release age setting, and teams should configure it in their projects. Safe Chain covers what those settings miss, such as a one-off npx command outside any project, or malicious packages older than the age limit.

Safe Chain covers only two ecosystems, but both have been targets of supply chain attacks. A version blocked for its age produces only a generic notice that packages were suppressed. Teams pulling through a private mirror with its own age limit gain little, and the malware blocking depends on a single vendor's feed. Safe Chain is licensed under the AGPL or a commercial license, so check the terms before building it into anything you distribute.

100. Swamp

Assess

Swamp is an automation tool built to be used by AI coding agents. When an agent carries out an operational task directly, it may do it differently each time and leave little record of what it did. With Swamp, the agent writes the automation instead, as TypeScript models of external systems and YAML workflows that chain the models' methods together. The workflows follow the same defined process every time, their definitions live in a Git repository and the output from each run is kept as versioned data the agent can query later. Credentials are injected from a vault at runtime, so they stay out of prompts. Models can also be packaged as extensions and shared through a registry. An agent instructed to write and commit scripts without Swamp can produce something similar, but without built-in typed interfaces, run history or reusable modules.

Swamp replaces System Initiative, which we featured in Assess in a previous volume. Swamp grew out of infrastructure management but is aimed at operational workflows more broadly, from incident response to data pipelines. The source is open under AGPLv3, although the official distribution uses a commercial license that requires a subscription for use at work. The project is new and moves fast, with a fresh release for every commit, so teams should try it on low-risk internal automation before depending on it.

101. Vale

Assess

Vale is a linter for prose. It parses Markdown, AsciiDoc and reStructuredText, skips code blocks and checks the text against style rules stored as YAML in the repository, so writing conventions can be enforced in the editor and in CI, much like code style. Packaged styles cover the Google, Microsoft and Red Hat documentation guides; others check for inclusive language or readability, and you can write your own. Vale has been part of docs-as-code toolchains for years, but the volume of LLM-drafted documentation, ADRs and PR descriptions makes it more relevant now. Packages such as vail-ai-tells flag the phrasing typical of generated text, and Vale works well as one of the feedback sensors for coding agents, allowing an agent to revise a draft before anyone reads it.

© Thoughtworks, Inc. All Rights Reserved.

Vale's checks are deterministic word and pattern matches, which can catch surface-level habits but miss weak reasoning or invented detail. Vale also depends largely on one maintainer, and our use so far has been by individuals working on their own writing rather than by client teams.

102. Varlock

Assess

Varlock is an open-source environment configuration and secrets-management layer designed to avoid exposing secrets to coding agents via plaintext. Instead of relying on informal `.env.example` conventions, Varlock introduces a committed `.env.schema` that describes variable names, types, validation rules and documentation. It separates the configuration contract from the actual values. Secrets are resolved and injected at runtime from external vaults such as 1Password, AWS Secrets Manager or Azure Key Vault. For workloads where an agent must call an authenticated service, Varlock also offers a credential proxy that provides a placeholder to the agent and substitutes the real value only at the network boundary. We've found this to be a useful pattern that replaces ad hoc helper scripts and extends secret injection across multiple environments without exposing credentials.

103. zizmor

Assess

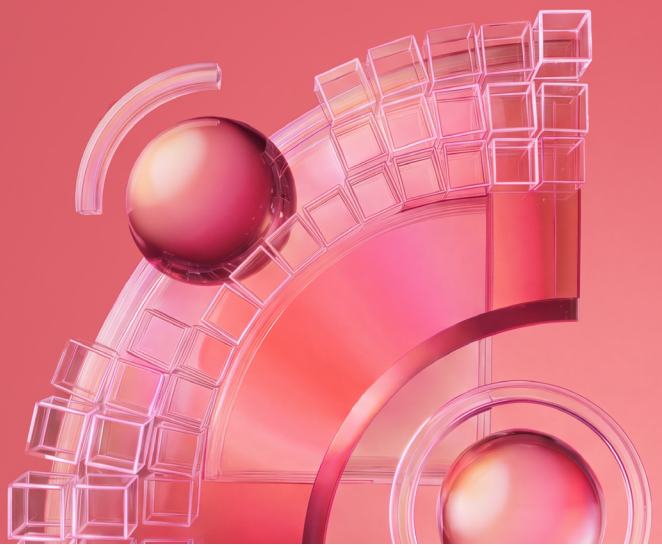
zizmor is a static analysis tool for GitHub Actions workflows that also checks Dependabot and pre-commit configurations. It flags misconfigurations attackers can exploit, including template injection, overly broad permissions and actions pinned to mutable tags as well as the use of risky triggers such as pull_request_target. It can also fix some findings automatically.

This year, attackers exploited a pull_request_target workflow in the Trivy repository, then rewrote the tags of its GitHub Action so pipelines referencing those tags ran the attackers' code. Our teams have used zizmor on client work, running it from the command line and as a pre-commit hook while writing workflows. Coding agents can run it too, providing quick feedback on the pipelines they write.

Amid the attention on AI-powered security scanning, we like having a deterministic tool for this job. Snyk and Wiz cover some of the same ground commercially and general linters catch some of these issues, but zizmor is free and open source. Its drawback is scope: it doesn't cover any other CI platforms beyond GitHub Actions. Teams using GitHub Actions should evaluate zizmor alongside zero trust security for CI/CD.

© Thoughtworks, Inc. All Rights Reserved.

Languages and Frameworks



Adopt

104. Open Data Contract Standard (ODCS)

Trial

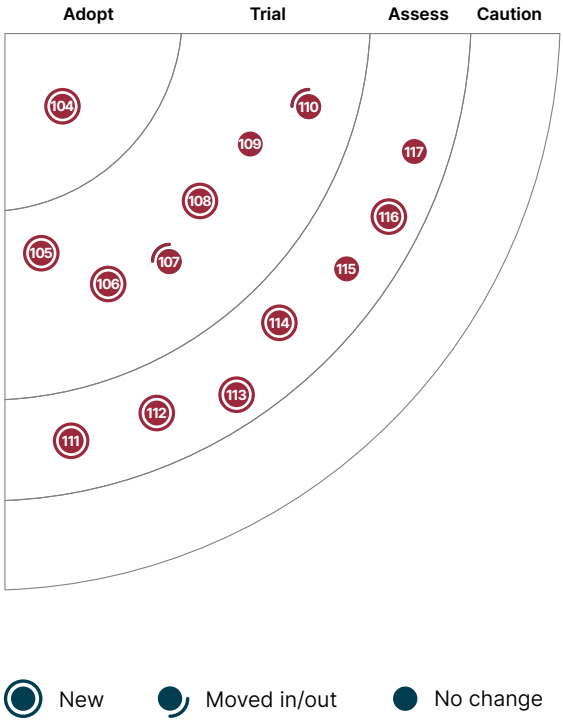
105. BoTorch
106. Deep Agents
107. Kagent
108. OpenFGA
109. Presidio
110. Promptfoo

Assess

111. Apple Core AI
112. CALM
113. Hindsight
114. Quint
115. SolidJS
116. spartan/ui
117. TLA+

Caution

—



104. Open Data Contract Standard (ODCS)

Adopt

The Open Data Contract Standard (ODCS) defines data contracts declaratively in YAML, covering a data product's schema, quality expectations and service levels. It's developed by the Linux Foundation's Bitol project, which graduated in July 2026. Although several competing standards and homegrown formats exist, ODCS has emerged as a leading standard, with the maintainers of Data Contract CLI deprecating DCS in favor of ODCS.

Several of our clients building data meshes and data products are moving to ODCS, often alongside the Open Data Product Standard (ODPS). Interest in data contracts is also growing, partly because their declarative format makes them easier for both humans and AI agents to interpret, review and validate. We're placing ODCS in Adopt as a sensible default for teams building data products.

105. BoTorch

Trial

BoTorch is an open-source, PyTorch-based framework for Bayesian optimization. While Bayesian optimization is often used for hyperparameter tuning, we're seeing it increasingly applied to real-world scientific and engineering problems. When standard tools for this challenge fall short, BoTorch provides the modularity and low-level control needed to construct probabilistic models and custom acquisition functions.

Paired with Ax for experiment orchestration, BoTorch excels in domains such as product formulation and R&D optimization, where physical or experimental evaluations are costly and exhaustive search is impractical. In these complex search spaces, BoTorch helps teams efficiently navigate trade-offs and maximize a limited evaluation budget.

106. Deep Agents

Trial

Deep Agents is an open-source agent harness from LangChain, built for long-running tasks. Deep Agents provides an opinionated architecture for breaking down problems, delegating work to subagents and managing execution. It includes key harness features such as planning tools, agent skills, memory, human-in-the-loop checkpoints and filesystem access.

Available as Python and TypeScript libraries, Deep Agents integrates well with LangChain and LangGraph. It's particularly useful when specific steps in a complex agentic graph require a capable, long-running agent. In our experience, it complements multi-agent research and legacy modernization use cases and integrates with the broader LangChain ecosystem, including observability tools like Langfuse.

107. Kagent

Trial

Kagent is an open-source programming framework and Kubernetes-native platform for building, running and managing AI agents. Instead of requiring bespoke infrastructure, the framework turns agents into standard Kubernetes custom resource definitions (CRDs) that teams can deploy, scale and secure using their existing GitOps pipelines. This allows teams proficient in Kubernetes to build

© Thoughtworks, Inc. All Rights Reserved.

custom agents without needing to learn a new operational model. Kagent provides a built-in runtime with OpenTelemetry, secure sandboxing, memory support and native Model Context Protocol (MCP) integrations to easily connect agents with external tools and model providers.

We've moved Kagent into Trial after successfully using it to support production environments. At a recent client, we transitioned a multi-agent support bot to Kagent from a previous tool that restricted connections to a single MCP server and required hosting on a personal account. Because the client's team was already familiar with Kubernetes, the migration was simple. The resulting deployment features an orchestrator and multiple specialist agents that use Kagent's built-in Agent-to-Agent (A2A) Protocol to communicate, helping developers debug incidents and solve general questions directly within team channels.

108. OpenFGA

Trial

OpenFGA is an open-source, fine-grained authorization system inspired by Google's Zanzibar. It models relationships between users and resources to determine permissions, enabling applications to answer questions such as "Can this user access this resource?"

Unlike traditional role-based access control (RBAC), OpenFGA supports complex, resource-specific permissions and hierarchical relationships. It centralizes authorization logic, improving maintainability, consistency, testing and auditing across distributed services.

We've successfully used OpenFGA in production for sophisticated relationship-based access control (ReBAC), deploying it with a sidecar strategy to enable low-latency permission checks. It's a compelling choice when standard RBAC becomes insufficient.

109. Presidio

Trial

Presidio is a data protection SDK for identifying and anonymizing sensitive data in structured and unstructured text. It detects personally identifiable information (PII) such as credit card numbers, names and locations using named entity recognition, regular expressions and rule-based logic. Presidio supports custom entity recognizers and de-identification pipelines, allowing organizations to tailor it to their privacy and compliance needs. Our teams have used Presidio in enterprise environments with strict data-sharing controls when integrating with LLMs. While it automates the detection of sensitive information, it isn't foolproof and may miss or misidentify entities. Teams should exercise caution when relying on its results.

110. Promptfoo

Trial

We put Promptfoo in Assess in 2023 for testing prompts. We still use it for evals, including agent tool calls and CI gates, but now also for red teaming. We recommend trialing it for both. To red-team an app, you describe what it's for, and Promptfoo generates attacks for the risks you choose, applies attack strategies and grades the responses. We used Promptfoo to red-team a production conversational assistant with domain-specific safety rules. It turned ad hoc manual probing into repeatable tests rerun after every fix. It also works with stateful guardrails: the guardrails are the

© Thoughtworks, Inc. All Rights Reserved.

runtime control, and Promptfoo tests that they hold. Promptfoo is now owned by OpenAI, although it remains MIT-licensed and is model agnostic. Some of Promptfoo's more advanced simulated attacks rely on its hosted service, so make sure you're comfortable with what data leaves your organization.

111. Apple Core AI

Assess

Apple Core AI is a recently launched framework for running AI and machine-learning models directly on devices using Apple Silicon. Providing native Swift support for mobile development, it acts as the modern successor to Apple's previous framework, Core ML. Apple Core AI allows developers to integrate AI features natively into their iOS applications, using on-device compute rather than relying on external services.

A particularly notable capability in this ecosystem is the Evaluations framework in Xcode, which extends Swift Testing to evaluate generative AI features built with Apple's Foundation Models. Evaluating nondeterministic LLM output has historically required custom back-end scripts or third-party AI evaluation tools. Now, teams can run synthetic datasets, measure quantitative metrics in code and apply model-as-a-judge scoring for qualitative criteria such as tone and relevance, all within native developer workflows.

112. CALM

Assess

The Common Architecture Language Model, or CALM, is an open-source specification from FINOS for defining software architectures in a standardized, machine-readable format. Built on an extensible JSON meta-schema, it treats architecture as code rather than relying on static diagrams and documentation that can drift from implementation.

In large enterprises, ensuring systems follow established design patterns can be challenging. CALM allows teams to define and manage architecture in a decentralized way while maintaining a unified view when needed. Its structured descriptions can also support validation against the intended architecture. While tools like Mermaid support diagrams as code, CALM provides a machine-readable foundation for architectural compliance and programmatic governance.

113. Hindsight

Assess

Hindsight is Vectorize's open-source memory system for AI agents. An LLM extracts facts, entities and timestamps from conversations and stores them in Postgres. Recall combines semantic, keyword, graph and time-based search and reranks the results. A reflect operation consolidates stored memories into observations and "mental models" that evolve as new evidence arrives. This consolidation sets it apart from Mem0, which extracts and stores the facts it judges most relevant. Hindsight maintains a separate memory bank for each user or agent and can be self-hosted or used through the managed Hindsight Cloud service. Agents connect to it through SDKs, framework integrations or an MCP server.

Hindsight is particularly useful for custom-built agents that need memory across sessions. One of our teams used it in a hackathon project to give agents per-user memory that improves over repeated sessions. Hindsight also has plugins for coding agents such as Claude Code, although those agents already have their own mechanism for retaining memory across sessions. The vendor reports leading

© Thoughtworks, Inc. All Rights Reserved.

scores on the LongMemEval benchmark, which tests recall over long chat histories. Teams building agents that need long-term memory for each user should compare Hindsight with Mem0 and measure whether its recall and consolidation capabilities improve agent behavior enough to justify the additional LLM calls and infrastructure.

114. Quint

Assess

Quint is a formal specification language based on the underlying calculus of TLA+, the temporal logic of actions. Like TLA+, it's designed to help you find subtle bugs in complex systems involving concurrency. Quint distinguishes itself from TLA+ through its readability and developer experience. It has a TypeScript-like syntax that makes review easier, especially when agents write the specification. It also has a type checker, REPL and simulator and can be installed with npm. The simulator exercises the model inexpensively with randomized runs and can also generate traces from the spec to test an implementation. Our teams have found the choice between Quint and TLA+ a close call, weighing Quint's tooling against TLA+'s maturity. We hope that its ongoing development signals growing momentum behind polyglot verification.

115. SolidJS

Assess

SolidJS is a declarative JavaScript library for creating user interfaces that continues to stand out for its high performance and fine-grained reactivity. Recent updates introduce significant built-in capabilities for asynchronous client-to-server communication. By handling these concerns natively, SolidJS eliminates the need for external libraries to manage API calls, caching and request deduplication. These features improve the developer experience by addressing common asynchronous challenges, such as race conditions and loading states, directly within the framework. SolidJS also provides detailed control over suspense boundaries, allowing developers to sequence how asynchronous components are revealed rather than leaving their order to network availability. Coupled with an expanding toolset that includes the SolidStart framework, SolidJS is a compelling alternative worth exploring for building high-performance web applications.

116. spartan/ui

Assess

spartan/ui applies the approach of shadcn to Angular. You install accessible, unstyled primitives built on the Angular CDK from npm and combine them with styled components that live in your own source code. This division is central to the design: accessibility, behavior and state are maintained by upstream libraries, while you control the styling.

It's a credible Angular Material alternative. One of our teams used it as the basis for a branded component library across several applications, avoiding the need to build a design system from scratch and reducing the effort required to build and maintain shared components. spartan/ui reached version 1.0 in June 2026, but it's still relatively young. We've encountered some rough edges when using the CLI to upgrade the styled components in our source code, so we're keeping it in Assess.

© Thoughtworks, Inc. All Rights Reserved.

117. TLA+

Assess

TLA+ is a formal specification language for identifying design flaws in distributed systems. When we last blipped it in 2015, the investment was too high for all but critical systems. Since then, coding agents have changed the calculus: the cost of using TLA+ has gone down, while its value has gone up. TLA+'s awkward developer experience is no longer a blocker, although teams looking for friendly syntax might prefer Quint. We've observed teams verify application traces against a TLA+ model without prior experience.

Meanwhile, developers now have greater need for quality assurance techniques to manage agent-generated code. Where TLA+ was once the preserve of organizations like AWS, which used it to find subtle bugs in DynamoDB, it's now part of a broader trend of polyglot verification, alongside techniques like mutation testing and property-based unit testing.

© Thoughtworks, Inc. All Rights Reserved.

Stay up to date with all Radar-related news and insights

Subscribe to the Technology Radar to receive emails every other month for tech insights from Thoughtworks and future Technology Radar releases.

Subscribe now



We are a global technology consultancy that delivers extraordinary impact by blending design, engineering and AI expertise.

For over 30 years, our culture of innovation and technology excellence has helped clients strengthen their enterprise systems, scale with agility and create seamless digital experiences.

We're dedicated to solving our clients' most critical challenges, combining AI and human ingenuity to turn their ambitious ideas into reality.

/thoughtworks

Design. Engineering. AI.