/thoughtworks

Strategy. Design. Engineering.

Volume 29 | September 2023

Technology Radar

An opinionated guide to today's technology landscape

About the Radar 3
Radar at a glance 4
Contributors 5
Themes 6
The Radar 8
Techniques 11
Platforms 20
Tools 26
Languages and Frameworks 39

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.

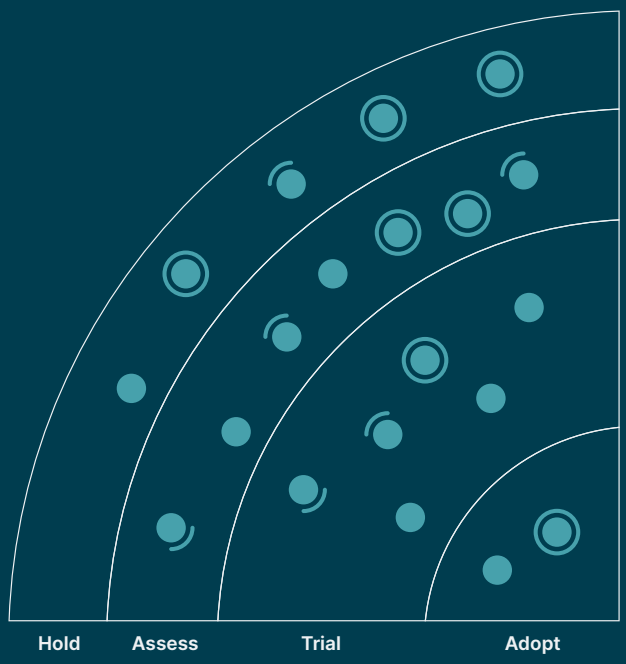
We encourage you to explore these technologies. The Radar is graphical in nature, grouping items into techniques, tools, platforms and languages and frameworks. When Radar items could appear in multiple quadrants, we chose the one that seemed most appropriate. We further group these items in four rings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

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

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' — their position in the Radar often changes — usually indicating 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.

Hold: Proceed with caution.

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.

Contributors

The Technology Advisory Board (TAB) is a group of 22 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, and CTO Emerita, Rebecca Parsons.

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 remotely in August 2023.

Rebecca Parsons (CTO Emerita)

Rachel Laycock (CTO)

Martin Fowler (Chief Scientist)

Bharani Subramaniam

Birgitta Böckeler

Brandon Byars

Camilla Falconi Crispim

Erik Dörnenburg

Fausto de la Torre

Hao Xu

Ian Cartwright

James Lewis

Marisa Hoenig

Maya Ormaza

Mike Mason

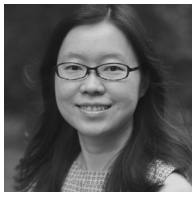
Neal Ford

Pawan Shah

Scott Shaw

Selvakumar Natesan

Shangqi Liu



Sofia Tania

Vanya Seth

Themes

AI-assisted software development

To no one's surprise, AI-related topics dominated our conversation for this edition of the Radar. For the first time ever, we needed a visual guide to untangle the different categories and capabilities (something we never had to resort to even in the heyday of chaos in the JavaScript ecosystem). As a software consultancy with a history of pioneering engineering practices like CI and CD, one of the categories of particular interest to us is using AI to assist in software development. As part of the Radar, we therefore discussed many coding assistance tools, like GitHub Copilot, Tabnine and Codeium. We're also excited about how open-source LLMs for coding might shake up the tooling landscape, and we see great promise in the explosion of tools and capabilities for assistance beyond coding as well, such as user story writeup assistance, user research, elevator pitches and other language-based chores. At the same time, we hope developers use all of these tools responsibly and stay firmly in the driver's seat, with things like hallucinated dependencies being just one of the security and quality risks to be aware of.

How productive is measuring productivity?

Software development can sometimes seem like magic to non-technologists, which leads managers to strive to measure just how productive developers are at their mysterious tasks. Our chief scientist, Martin Fowler, wrote about this topic as long ago as 2003, but it hasn't gone away. We discussed many modern tools and techniques for this Radar that take more nuanced approaches to measuring the creative process of building software yet still remain inadequate. Fortunately, the industry has moved away from using lines of code as a measure of output. However, alternative ways to measure the A ("Activity") of the SPACE framework, such as number of pull requests or issues resolved, are still poor indicators of productivity. Instead, the industry has started focusing on *engineering effectiveness*: rather than measure productivity, we should measure things we know contribute to or detract from the flow. Instead of focusing on an individual's activities, we should focus on the sources of waste in the system and the conditions we can empirically show have an impact on the developer's perception of "productivity." New tools such as DX DevEx 360 address this by focusing on the developer experience rather than some specious measure of output. However, many leaders continue to refer to developer "productivity" in a vague, qualitative way. We suspect that at least some of this resurgence of interest concerns the impact of AI-assisted software development, which raises the inevitable question: is it having a positive impact? While measurements may be gaining some nuance, real measurements of productivity are still elusive.

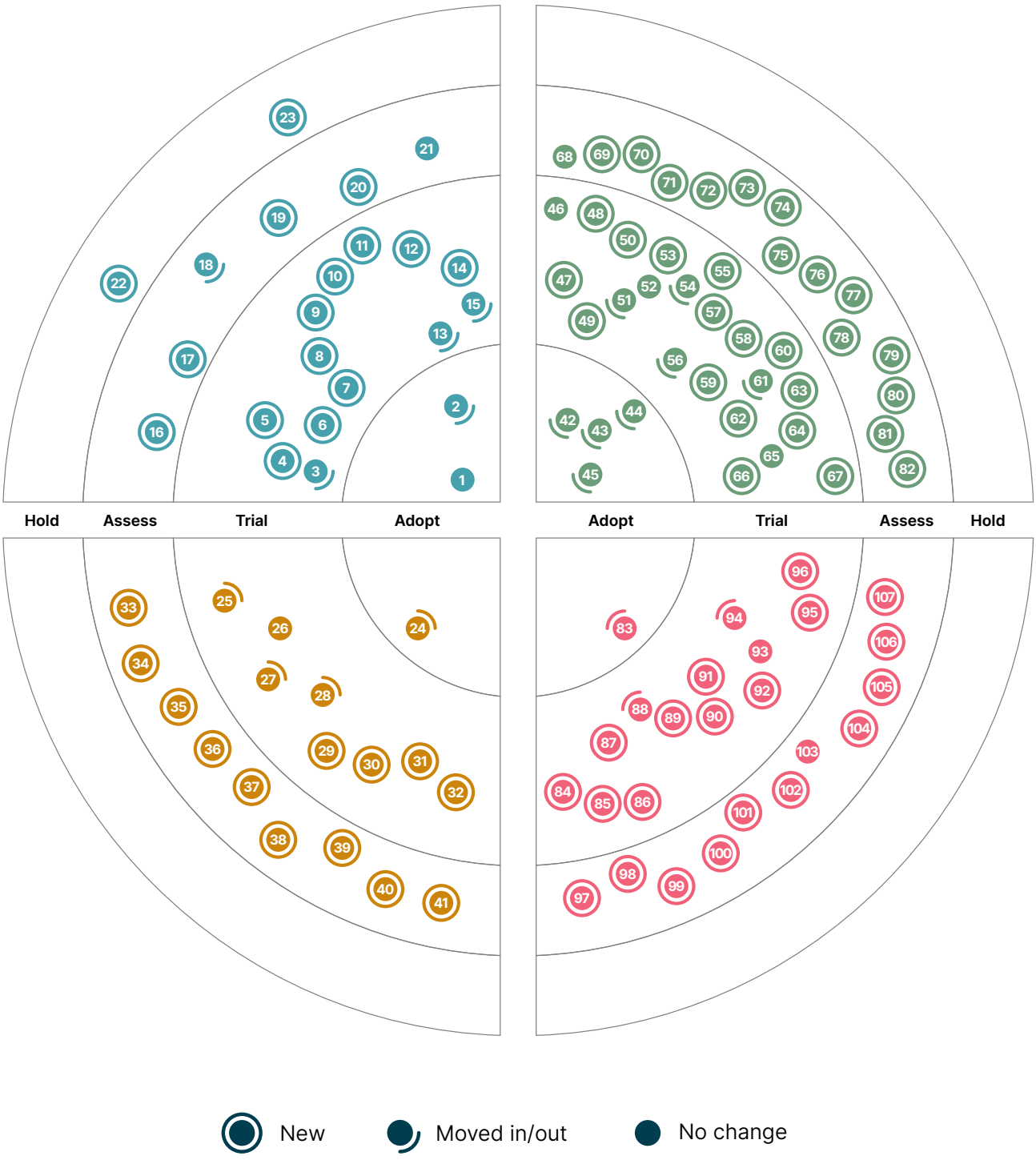
A large number of LLMs

Large language models (LLMs) form the basis for many modern breakthroughs in AI. Much current experimentation involves prompting chat-like user interfaces such as ChatGPT or Bard. Fundamentally, the core competing ecosystems (OpenAI's ChatGPT, Google's Bard, Meta's LLaMA, Amazon's Bedrock among others) featured heavily in our discussions. More broadly, LLMs are tools that can solve a variety of problems, ranging from content generation (text, images and videos) to code generation to summarization and translation, to name a few. With natural language serving as a powerful abstraction layer, these models present a universally appealing tool set and are therefore being used by many information workers. Our discourse encompasses various facets of LLMs, including self-hosting, which allows customization and greater control than cloud-hosted LLMs. With the growing complexity of LLMs, we deliberate on the ability to quantize and run them on small form factors, especially in edge devices and constrained environments. We touch upon ReAct prompting, which holds promise for improved performance, along with LLM-powered autonomous agents that can be used to build dynamic applications that go beyond question and answer interactions. We also mention several vector databases (including Pinecone) that are seeing a resurgence thanks to LLMs. The underlying capabilities of LLMs, including specialized and self-hosted capabilities, continues its explosive growth.

Remote delivery workarounds mature

Even though remote software development teams have leveraged technology to overcome geographic constraints for years now, the pandemic's impact fueled innovation in this area, solidifying full remote or hybrid work as an enduring trend. For this Radar, we discussed how remote software development practices and tools have matured, and teams keep pushing boundaries with a focus on effective collaboration in an environment that is more distributed and dynamic than ever. Teams keep coming up with innovative solutions using new collaborative tools. Others continue to adapt and improve existing in-person practices for activities like real-time pair programming or mob programming, distributed workshops (e.g., remote Event Storming) and both asynchronous and synchronous communication. Although remote work offers numerous benefits (including a more diverse talent pool), the value of face-to-face interactions is clear. Teams shouldn't let critical feedback loops lapse and need to be aware of the trade-offs they incur when transitioning to remote settings.

The Radar



The Radar

Techniques

Adopt

1. Design systems
2. Lightweight approach to RFCs

Trial

3. Accessibility-aware component test design
4. Attack path analysis
5. Automatic merging of dependency update PRs
6. Data product thinking for FAIR data
7. OIDC for GitHub Actions
8. Provision monitors and alerts with Terraform
9. ReAct prompting
10. Retrieval-Augmented Generation (RAG)
11. Risk-based failure modeling
12. Semi-structured natural language for LLMs
13. Tracking health over debt
14. Unit testing for alerting rules
15. Zero trust security for CI/CD

Assess

16. Dependency health checks to counter package hallucinations
17. Design system decision records
18. GitOps
19. LLM-powered autonomous agents
20. Platform orchestration
21. Self-hosted LLMs

Hold

22. Ignoring OWASP Top 10 lists
23. Web components for server-side-rendered (SSR) web apps

Platforms

Adopt

24. Colima

Trial

25. CloudEvents
26. DataOps.live
27. Google Cloud Vertex AI
28. Immuta
29. Lokalise
30. Orca
31. Trino
32. Wiz

Assess

33. ActivityPub
34. Azure Container Apps
35. Azure OpenAI Service
36. ChatGLM
37. Chroma
38. Kraftful
39. pgvector
40. Pinecone
41. wazero

Hold

—

Tools

Adopt

42. dbt
43. Mermaid
44. Ruff
45. Snyk

Trial

46. AWS Control Tower
47. Bloc
48. cdk-nag
49. Checkov
50. Chromatic
51. Cilium
52. Cloud Carbon Footprint
53. Container Structure Tests
54. Devbox
55. DX DevEx 360
56. GitHub Copilot
57. Insomnia
58. IntelliJ HTTP Client plugin
59. KEDA
60. Kubeconform
61. mob
62. MobSF
63. Mocks Server
64. Prisma runtime defense
65. Terratest
66. Thanos
67. Yalc

Assess

68. ChatGPT
69. Codeium
70. GitHub merge queue
71. Google Bard
72. Google Cloud Workstations
73. Gradio
74. KWOK
75. Llama 2
76. Maestro
77. Open-source LLMs for coding
78. OpenCost
79. OpenRewrite
80. OrbStack
81. Pixie
82. Tabnine

Hold

—

Languages and Frameworks

Adopt

83. Playwright

Trial

84. .NET Minimal API
85. Ajv
86. Armeria
87. AWS SAM
88. Dart
89. fast-check
90. Kotlin with Spring
91. Mockery
92. Netflix DGS
93. OpenTelemetry
94. Polars
95. Pushpin
96. Snowpark

Assess

97. Baseline Profiles
98. GGML
99. GPTCache
100. Grammatical Inflection API
101. htmx
102. Kotlin Kover
103. LangChain
104. LlamaIndex
105. promptfoo
106. Semantic Kernel
107. Spring Modulith

Hold

—

Techniques

Adopt

1. Design systems
2. Lightweight approach to RFCs

Trial

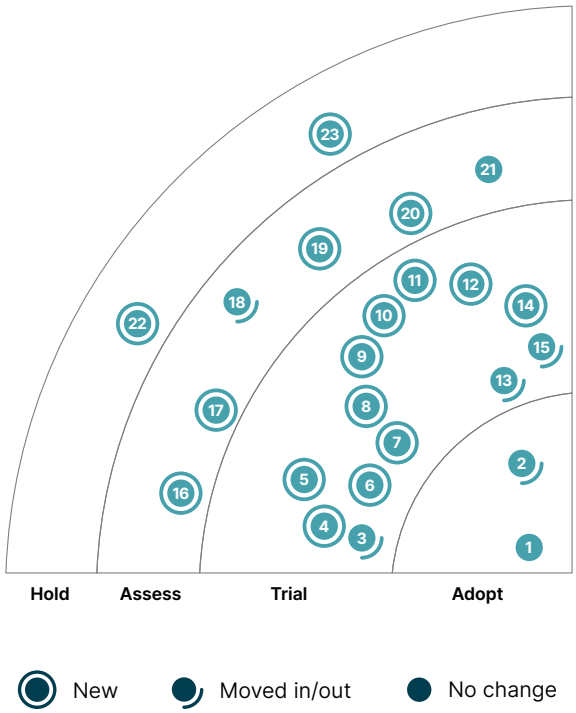
3. Accessibility-aware component test design
4. Attack path analysis
5. Automatic merging of dependency update PRs
6. Data product thinking for FAIR data
7. OIDC for GitHub Actions
8. Provision monitors and alerts with Terraform
9. ReAct prompting
10. Retrieval-Augmented Generation (RAG)
11. Risk-based failure modeling
12. Semi-structured natural language for LLMs
13. Tracking health over debt
14. Unit testing for alerting rules
15. Zero trust security for CI/CD

Assess

16. Dependency health checks to counter package hallucinations
17. Design system decision records
18. GitOps
19. LLM-powered autonomous agents
20. Platform orchestration
21. Self-hosted LLMs

Hold

22. Ignoring OWASP Top 10 lists
23. Web components for server-side-rendered (SSR) web apps



1. Design systems

Adopt

As application development becomes increasingly dynamic and complex, it's a challenge to deliver accessible and usable products with consistent style. This is particularly true in larger organizations with multiple teams working on different products. Design systems define a collection of design patterns, component libraries and good design and engineering practices that ensure consistent digital products. Evolved from the corporate style guides of the past, design systems offer shared libraries and documents that are easy to find and use. Generally, guidance is written down as code and kept under version control so that the guide is less ambiguous and easier to maintain than simple documents. Design systems have become a standard approach when working across teams and disciplines in product development because they allow teams to focus. They can address strategic challenges around the product itself without reinventing the wheel every time a new visual component is needed.

Our experiences show that teams seldom apply a product-centric mindset when building design systems. The primary consumers of the shared libraries and documents are the product development teams. When applying a product mindset, system owners should establish empathy with internal consumers (the development teams) and collaborate with them. We've found that the reason many component libraries are maligned is because the owning team wasn't able to give consumers what they needed fast enough and wasn't set up to take outside contributions. A product-centric mindset also requires organizations to think about if and how contributions should be made to the design system and how these contributions should be governed — on this topic, we recommend applying the design system decision records technique. For us, running a good design system or component library requires social work as much as technical work.

2. Lightweight approach to RFCs

Adopt

A Request for Comments (RFC) is a formal document that includes context-dependent design and architectural ideas to facilitate team collaboration and decision-making. Nearly all digital native and scaleup organizations use RFCs to capture decisions around design, architecture, techniques and the ways their teams collaborate. Mature organizations have used RFCs in autonomous teams to drive better communication and collaboration, especially in cross-team decision-making. They're often used as a process to review and ratify architecture decision records. The result is a transparent collaborative process that allows those affected by a decision the chance to weigh in and provide input before the decision is ratified. So often in fast-moving environments, the reasoning leading up to design decisions gets lost along the way and teams who are responsible for implementing the decision are left scratching their heads. An RFC provides a decision audit record that benefits future team members and captures the technical and business evolution of an organization. An RFC can be a valuable tool for facilitating evolutionary architecture. For the best outcome, though, we recommend taking a *lightweight* approach to RFCs. If not narrowly scoped and to the point, these documents tend to grow in length over time and start resembling traditional solution architecture documents that are filed away and forgotten.

3. Accessibility-aware component test design

Trial

One of the many places in the software delivery process where accessibility requirements should be considered is during web component testing. While testing framework plugins like chai-a11y-axe provide assertions in their API to check for the basics, accessibility-aware component test design can further help to provide all the semantic elements screen readers and other assistive technologies require. First, instead of using test ids or classes to find and select the elements you want to validate, use a principle of identifying elements by ARIA roles or other semantic attributes that are used by assistive technologies. Some testing libraries, like Testing Library, even recommend this in their documentation. Second, do not just test for click interactions; also consider people who cannot use a mouse or see the screen, and consider adding additional tests for the keyboard and other interactions. The described technique is well established within our teams, and we should have placed it in the Trial ring a while ago.

4. Attack path analysis

Trial

Attack path analysis is a security analysis technique that identifies and assesses the potential paths that an attacker could take to exploit vulnerabilities in an organization's systems and networks. Previously most security analysis strategies or tools have focused on particular risk areas like misconfigurations, vulnerable containers or CVE alerts. This siloed approach means that teams cannot see how risks can be combined with weaknesses in other layers of the technology stack to create dangerous attack paths. Although this technique isn't new, recent advancements in security analysis tools have made it more accessible to security teams. Orca and Wiz are two such tools. We suggest teams managing complex infrastructures consider this technique while planning a security strategy or selecting the security analysis tools for their organization.

5. Automatic merging of dependency update PRs

Trial

The complexity of the software supply chain is a major risk, and we've covered it extensively, for example, in our write-ups on SBOM and SLSA. The Achilles heel for most teams is still the presence of vulnerabilities in dependencies, often indirect dependencies several levels down. Tools such as Dependabot help by creating pull requests (PRs) to update dependencies. It requires engineering discipline, though, to look after these PRs promptly, especially when they are for applications or services that are not under active development.

Under the right circumstances we now advocate for automatic merging of dependency update PRs. This requires that the system has extensive test coverage — not only unit tests but also functional and performance tests. The build pipeline must run all of these tests, and it must include security scanning. In short, the team must have full confidence that when the pipeline runs successfully the software is ready to go into production. In such cases, dependency update PRs, even when they include major version updates in indirect dependencies, should be merged automatically.

6. Data product thinking for FAIR data

Trial

Data product thinking prioritizes treating data consumers as customers, ensuring they have a seamless experience across the data value chain. This encompasses ease of data discovery, understanding, trust, access and consumption. "Product thinking" is not a new concept. In the past we've embraced this in the operational world while building operational products or microservices. It also suggests a new way to build long-lived cross-functional teams to own and share data across the organization. By bringing a product mindset to data, we believe organizations can operationalize the FAIR (findable, accessible, interoperable and reusable) principles. Our teams use data catalogs such as Collibra and DataHub to enable data product discoverability. To foster trust, we publish data quality and SLI metrics like freshness, completeness, consistency for each data product, and tools such as Soda Core and Great Expectations automate the data quality checks. Data Observability, meanwhile, can be achieved with the help of platforms like Monte Carlo.

We've seen data products evolve as the reusable building blocks for multiple use cases over a period of time. This is accompanied by faster time to market for subsequent use cases as we progress on identifying and building value case-driven data products. Hence, our advice is to embrace data product thinking for FAIR data.

7. OIDC for GitHub Actions

Trial

One of the techniques we recommend for implementing zero trust security for CI/CD is to authenticate your pipelines for cloud services access via federated identity mechanisms like OpenID Connect (OIDC). As GitHub Actions is widely used — and this important technique remains underused — we want to call out OIDC for GitHub Actions. This way you can avoid storing long-lived access tokens for your cloud resources, and your pipelines won't get direct access to secrets. However, be sure to scope access carefully so that actions really run with least privilege.

8. Provision monitors and alerts with Terraform

Trial

Infrastructure as code (IaC) is now a widely accepted approach for defining and provisioning hosting environments. Even with the continual evolution of tools and techniques in this area, Terraform continues to be the dominant tool for doing IaC on cloud-native resources. However, most hosting environments today are complex combinations of cloud vendor–native services, third-party services and custom code. In these environments, we've found that engineers often resort to a mixture of Terraform for cloud resources and custom scripts for the rest. This can lead to a lack of consistency and repeatability in the provisioning process. In fact, many of the third-party services that are commonly used in hosting environments — including Splunk, Datadog, PagerDuty and New Relic — have Terraform providers that you can use to provision and configure these services. That's why we recommend that, in addition to cloud resources, teams also provision monitors and alerts with Terraform. This leads to IaC with better modularity that is easier to understand and maintain. As with all IaC, there is a risk of introducing inconsistencies when the configuration is changed via other interfaces. To ensure that the Terraform code remains the source of truth, we recommend you disable configuration changes via user interfaces and APIs.

9. ReAct prompting

Trial

ReAct prompting is a method for prompting LLMs intended to improve the accuracy of their responses over competing methods such as chain-of-thought (CoT). Introduced in a 2022 paper, it works by bringing together reasoning and action (hence ReAct). Such an approach helps make LLM responses more explainable and reduces hallucinations compared to CoT, giving prompters a better chance of getting what they want. LangChain was originally developed to support this style of prompting. Autonomous agents based on ReAct prompting have proven to be some of the most widely used applications of LLMs our teams have been building. Recently, OpenAI introduced function calling to its APIs to make ReAct and similar prompting styles easier to implement without resorting to external tools like LangChain. We're still in the early stages of defining this discipline, but so far, ReAct and its descendants have pointed the way to some of the most exciting applications of LLMs.

10. Retrieval-Augmented Generation (RAG)

Trial

Retrieval-Augmented Generation (RAG) is a technique to combine pretrained parametric and nonparametric memory for language generation. It enables you to augment the existing knowledge of pretrained LLMs with the private and contextual knowledge of your domain or industry. With RAG, you first retrieve a set of relevant documents from the nonparametric memory (usually via a similarity search from a vector data store) and then use the parametric memory of LLMs to generate output that is consistent with the retrieved documents. We find RAG to be an effective technique for a variety of knowledge intensive NLP tasks — including question answering, summarization and story generation.

11. Risk-based failure modeling

Trial

Risk-based failure modeling is a process used to understand the impact, likelihood and ability of detecting the various ways that a system can fail. Delivery teams are starting to use this methodology to design and evaluate the controls needed to prevent these failures. The approach is based on the practice of failure modes and effects analysis (FMEA), a risk-scoring technique that has been around since the 1940s and with a successful track record in industries that build complex physical systems such as aerospace and automotive. As with those industries, software failure can also have dire consequences — compromising, for example, human health and privacy — which is why we're seeing an increased need for systems to undergo rigorous analysis. The process starts by identifying the possible failure modes. The team then performs a root cause analysis and assigns scores according to the likelihood of a failure occurring, the size of its impact and the probability of detecting the root cause of the failure. We've found this to be most effective when cross-functional teams iterate through this process as the system evolves. When it comes to security, risk-based failure modeling can be a useful complement to threat modeling and attack path analysis.

12. Semi-structured natural language for LLMs

Trial

We've had success in various applications using a semi-structured natural language for LLMs. Structured inputs, such as a JSON document, are clear and precise and give the model an indication of the type of response being sought. Constraining the response in this way helps narrow the problem space and can produce more accurate completions, particularly when the structure conforms to a

domain-specific language (DSL) whose syntax or schema is provided to the model. We've also found that augmenting the structured input with natural language comments or notations produces a better response than either natural language or structured input alone. Typically, natural language is simply interspersed with structured content when constructing the prompt. As with many LLM behaviors, we don't know exactly why this works, but our experience shows that putting natural language comments in human-written code also improves the quality of output for LLM-based coding assistants.

13. Tracking health over debt

Trial

We keep experiencing the improvements teams make to their ecosystem by treating the health rating the same as other service-level objectives (SLOs) and prioritizing enhancements accordingly, instead of solely focusing on tracking technical debt. By allocating resources efficiently to address the most impactful issues related to health, teams and organizations can reduce long-term maintenance costs and evolve products more efficiently. This approach also enhances communication between technical and nontechnical stakeholders, fostering a common understanding of the system's state. Although metrics may vary among organizations (see this blog post for examples) they ultimately contribute to long-term sustainability and ensure software remains adaptable and competitive. In a rapidly changing digital landscape, focusing on tracking health over debt of systems provides a structured and evidence-based strategy to maintain and enhance them.

14. Unit testing for alerting rules

Trial

Observability and monitoring are essential for software teams. Given the unpredictable nature of certain events, creating accurate alert mechanisms with complex rules is crucial. However, the true validation of these rules comes only when scenarios arise in the wild. The unit testing for alerting rules technique allows teams to better define rules by proactively testing and refining them beforehand, increasing confidence in the way the rule is set up. This helps to reduce false alarms and ensure genuine issues are highlighted. Tools like Prometheus support unit testing for rules; our teams are already reporting its benefits in real-world settings.

15. Zero trust security for CI/CD

Trial

If not properly secured, the infrastructure and tools that run our build and delivery pipelines can become a big liability. Pipelines need access to critical data and systems like source code, credentials and secrets to build and deploy software. This makes these systems very inviting to malicious actors. We therefore highly recommend applying zero trust security for CI/CD pipelines and infrastructure — trusting them as little as necessary. This encompasses a number of techniques: If available, authenticate your pipelines with your cloud provider via federated identity mechanisms like OIDC, instead of giving them direct access to secrets; implement the principle of least privilege by minimizing the access of individual user or runner accounts, rather than employing "god user accounts" with unlimited access; use your runners in an ephemeral way instead of reusing them, to reduce the risk of exposing secrets from previous jobs or running jobs on compromised runners; keep the software in your agents and runners up to date; monitor the integrity, confidentiality and availability of your CI/CD systems the same way you would monitor your production software.

We're seeing teams forget about these types of practices, particularly when they're used to working with a self-managed CI/CD infrastructure in internal network zones. While all of these practices are important in your internal networks, they become even more crucial when using a managed service, as that extends the attack surface and blast radius even more.

16. Dependency health checks to counter package hallucinations

Assess

Securing the software supply chain has become a common concern among delivery teams, reflected in the growing number of tools and techniques in the space, several of which we've covered previously in the Radar. The growing popularity of GenAI-based tools as aids to the software development process has introduced a new software supply chain attack vector: package hallucinations. We believe it's important for teams that use such GenAI tools in their development process to stay vigilant against this risk. To do so, teams can perform dependency health checks to counter package hallucination: look at the date it was created, download numbers, comments and stars, the number of contributors, activity history and so on before choosing to adopt. Some of these can be performed on package repositories and GitHub, and tools like deps.dev and Snyk advisor can also provide additional input. Although it's not a new technique, it's gaining renewed relevance as teams increasingly experiment with GenAI tools in their software development process.

17. Design system decision records

Assess

In a fast-paced product development environment where users' needs constantly evolve, design is an area that is ever-changing. This means input on design decisions will continue to be required. Borrowing from the idea of documenting architecture decisions via ADRs, we started adopting a similar format — design system decision records — in order to document design system decisions with the corresponding rationale, research insights and experiment results. Communicating design system decisions effectively seems to be an emerging need of product development teams; doing it in this light manner is also recommended by zeroheight. This technique helped us reduce onboarding times, move conversations forward and align work streams that share the same design system.

18. GitOps

Assess

GitOps is a technique for deploying applications via the control loop pattern. An operator keeps the deployed application synchronized with configuration, usually a Git repository. When we last wrote about GitOps, the community had yet to agree on a definition of the term. At the time, we were concerned about common interpretations of the technique that included approaches like "branch per environment" for configuration, which may lead to snowflakes as code. Moreover, the messaging around GitOps as an alternative to continuous delivery was confusing. Since then, the four GitOps principles have clarified the scope and nature of the technique. When you peel away the hype and confusion, GitOps is a useful technique that takes advantage of the functionality of a Kubernetes cluster and creates opportunities to separate concerns between configuring an application and the implementation of the deployment process. Some of our teams have implemented GitOps as part of their continuous delivery setup with positive experiences, which is why we recommend assessing it.

19. LLM-powered autonomous agents

Assess

As development of large language models continues, interest in building autonomous AI agents is strong. AutoGPT, GPT-Engineer and BabyAGI are all examples of LLM-powered autonomous agents that drive an underlying LLM to understand the goal they have been given and to work toward it. The agent remembers how far it has progressed, uses the LLM in order to reason about what to do next, takes actions and understands when the goal has been met. This is often known as chain-of-thought reasoning — and it can actually work. One of our teams implemented a client service chatbot as an autonomous agent. If the bot cannot achieve the customer's goal, it recognizes its own limitation and redirects the customer to a human instead. This approach is definitely early in its development cycle: autonomous agents often suffer from a high failure rate and incur costly AI service fees, and at least one AI startup has pivoted away from an agent-based approach.

20. Platform orchestration

Assess

With the widespread adoption of platform engineering, we're seeing a new generation of tools that go beyond the traditional platform-as-a-service (PaaS) model and offer published contracts between developers and platform teams. The contract might involve provisioning cloud environments, databases, monitoring, authentication and more in a different environment. These tools enforce organizational standards while granting developers self-service access to variations through configuration. Examples of these platform orchestration systems include Kratix and Humanitec Platform Orchestrator. We'd recommend platform teams assess these tools as an alternative to pulling together your own unique collection of scripts, native tools and infrastructure as code. We've also noted a similarity to the concepts in the Open Application Model (OAM) and its reference orchestrator KubeVela, although OAM claims to be more application-centric than workload-centric.

21. Self-hosted LLMs

Assess

Large language models (LLMs) generally require significant GPU infrastructure to operate, but there has been a strong push to get them running on more modest hardware. Quantization of a large model can reduce memory requirements, allowing a high-fidelity model to run on less expensive hardware or even a CPU. Efforts such as llama.cpp make it possible to run LLMs on hardware including Raspberry Pis, laptops and commodity servers.

Many organizations are deploying self-hosted LLMs. This is often due to security or privacy concerns, or, sometimes, a need to run models on edge devices. Open-source examples include GPT-J, GPT-JT and Llama. This approach offers better control of the model in fine-tuning for a specific use case, improved security and privacy as well as offline access. Although we've helped some of our clients self-host open-source LLMs for code completion, we recommend you carefully assess the organizational capabilities and the cost of running such LLMs before making the decision to self-host.

22. Ignoring OWASP Top 10 lists

Hold

The OWASP Top 10 has long been a go-to reference for the most critical security risks to web applications. Despite being well-known, we've previously written about it being underused in the software development process and cautioned against ignoring OWASP Top 10.

What is less well-known is that OWASP also publishes similar top 10 lists for other categories. The OWASP Top 10 list for LLMs, whose first major version was released early August, highlights risks such as prompt injection, insecure output handling, training data poisoning and others that individuals and teams building LLM applications would do well to be aware of. OWASP has also recently released the second version of its OWASP Top 10 list for APIs. Given the OWASP Top 10 lists' breadth of coverage (web applications, APIs, LLMs and more), quality and relevance to the continuously changing security landscape, we extend our previous recommendation to caution teams against ignoring OWASP Top 10 lists.

23. Web components for server-side-rendered (SSR) web apps

Hold

Since we first mentioned them in 2014, web components have become popular, and, on the whole, our experience has been positive. Similarly, we've voiced our support for rendering HTML on the server by cautioning against SPA by default and by including frameworks such as Next.js and HTMX in addition to traditional server-side frameworks. However, although it's possible to combine both, it can also prove deeply problematic; that's why we suggest avoiding web components for server-side-rendered (SSR) web apps. As a browser technology, it's not trivial to use web components on the server. Frameworks have sprung up to make this easier, sometimes even using a browser engine, but the complexity is still there. Worse than the issues with developer experience is the user experience: Page load performance is impacted when custom web components have to be loaded and hydrated in the browser, and even with pre-rendering and careful tweaking of the component, a "flash of unstyled content" or some layout shifting is all but unavoidable. The decision to forgo web components can have far-reaching consequences as one of our teams experienced when they had to move their design system away from the web components–based Stencil.

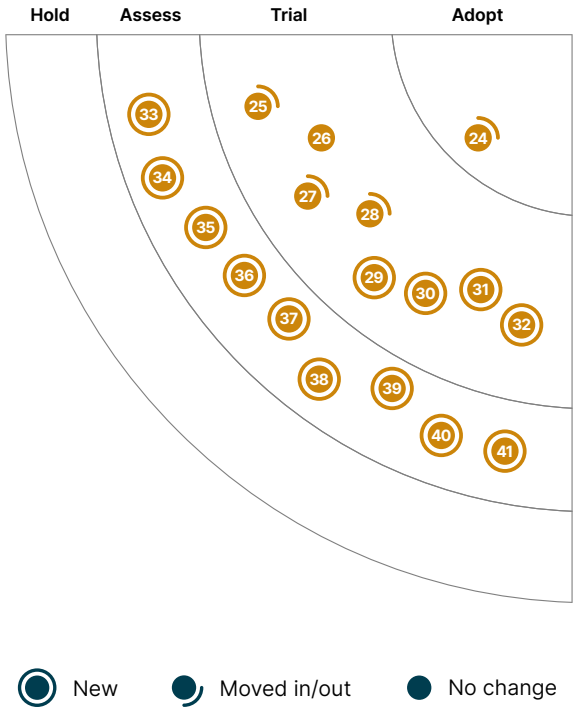
Platforms

Adopt
24. Colima

Trial
25. CloudEvents
26. DataOps.live
27. Google Cloud Vertex AI
28. Immuta
29. Lokalise
30. Orca
31. Trino
32. Wiz

Assess
33. ActivityPub
34. Azure Container Apps
35. Azure OpenAI Service
36. ChatGLM
37. Chroma
38. Kraftful
39. pgvector
40. Pinecone
41. wazero

Hold
—



24. Colima

Adopt

Colima is now our go-to alternative to Docker Desktop on macOS. We continue to use it on several projects to provision the Docker container run time in a Lima VM, to configure the Docker CLI on macOS and to handle port-forwarding and volume mounts. Colima can be configured to run containerd as its run time, which is also the run time on most managed Kubernetes services, improving the important dev-prod parity.

25. CloudEvents

Trial

Events are common mechanisms in event-driven architecture or serverless applications. However, producers or cloud providers tend to support them in different forms, which prevents interoperability across platforms and infrastructures. CloudEvents is a specification for describing event data in common formats to provide interoperability across services, platforms and systems. It provides SDKs in multiple languages so you can embed the spec into your application or toolchain. Our teams use it not only for cross-cloud platform purposes but also for domain event specification, among other scenarios. CloudEvents is hosted by the Cloud Native Computing Foundation (CNCF) and now runs as an incubator project that has been gaining increasing industry attention.

26. DataOps.live

Trial

DataOps.live is a data platform that automates environments in Snowflake. Inspired by DevOps practices, DataOps.live lets you treat the data platform like any other web platform by embracing continuous integration and continuous delivery (CI/CD), automated testing, observability and code management. Our teams are using it for managing the lifecycle of data products that includes development, branching and deployment of both code and data. With its automated environment management, it's very easy to build environments based on feature branches, modify them and destroy them automatically. It's also worth noting its declarative specification (SOLE) capability, which enables a streamlined developer experience. This allows teams to reduce the time it takes to build data products from months to days. Our teams have been using DataOps.live successfully in production, and that's why we recommend this platform when working with Snowflake.

27. Google Cloud Vertex AI

Trial

Significant developments have happened in the AI landscape since we first blipped Google Cloud Vertex AI. Since May 2023, Google has introduced several services and features to enrich this realm. These additions include Model Garden, a repository of 100+ pre-trained models; Generative AI Studio, a console intended to rapidly explore and prototype generative AI models; and Vertex AI Extensions which provides fully managed developer tools to connect AI models and real-time data or actions via APIs. The platform has evolved to offer GenAI models and integration support, and we're excited to use it more extensively.

28. Immuta

Trial

Since we last wrote about Immuta, our teams have gained significant experience with this data security platform. Its highlights include the ability to define subscription and data policies as code, version control and the ability to deploy these policies automatically to higher environments. Its ABAC support allows us to associate tags to data sources; if the same tag is associated with the user, access is granted. By leveraging Immuta and Snowflake integration we've been able to automate granting access to data products or data sets in a self-serve fashion. When the "user" requests access to a data product or a data set, the data product tag is then associated with the "user" as an attribute upon approval. Since the attribute on the "user" matches the tag on the data source, access is granted automatically courtesy of Immuta's Global Subscription policy. It's also worth noting Immuta's data masking policies which preserve data privacy by masking and restricting PII information to a specific user. Additional access to sensitive information at a much more granular level can be defined using row-level security policies that ensure users only have access to the specific data they're authorized to view. We've been happy with Immuta which is why we're moving it to Trial — it provides a good developer experience and makes it easier for large organizations to manage data policies.

29. Lokalise

Trial

Lokalise is a fully automated localization platform that allows for context-specific translations. Our teams use the Lokalise API in their ETL pipelines or development workflows to translate localizable information. Lokalise supports multiple file formats for the localizable strings. One aspect to highlight is the ability to upload an entire file, where each key-value pair is treated as a separate record and translated. Under the hood we leveraged Lokalise's integration with Google MT to take care of the translations. The Lokalise web UI provides ease of access to human reviewers to verify the translations, shorten them and rephrase them as they deem fit. In the past we've highlighted similar tools such as Phrase. Our teams have had a good experience with Lokalise, and we recommend you evaluate the platform for collaborative translation workflows.

30. Orca

Trial

Orca is a proprietary cloud security platform that identifies, prioritizes and remediates security risks and compliance issues. It supports major cloud providers and hybrid setups. Orca has extensive security queries/rules to continuously monitor deployed workloads for misconfigurations, vulnerabilities and compliance issues. It supports cloud VMs, serverless functions, containers and Kubernetes applications for the deployed workloads. These inbuilt security rules are consistently updated to keep pace with the evolving compliance standards and threat vectors. Since Orca is agentless, it offers a good developer experience and is easy to set up. Another notable feature is that it facilitates shift left security. Our teams use Orca CLI for scanning container images and IaC templates for vulnerabilities and misconfigurations as a pre-commit hook or as part of CI/CD workflows. It also continuously monitors and scans container registries (e.g., AWS ECR) for vulnerable base images or weak OS dependencies for already published images. Based on our teams' experiences, Orca provides a unified view of the security posture across the path to production, and for that reason we place it in Trial.

31. Trino

Trial

Trino, previously known as PrestoSQL, is an open-source, distributed SQL query engine designed for interactive analytic queries over big data. It is optimized to run both on-premise and in the cloud. It supports querying data where it lives, including Hive, Cassandra, relational databases and even proprietary data stores. For authentication mechanisms, it supports password-based authentication, LDAP and OAuth. For authorization and access control, Trino provides the ability to grant access at the catalog, schema and table levels. Our teams used resource groups spliced according to consumption patterns like visualization, reporting or machine learning use cases to manage and limit resource usage. The JMX-based monitoring provides a rich set of metrics to enable cost attribution at query or user level. Our teams use Trino as a gateway for data access across a variety of sources. When it comes to querying extremely large-scale data, Trino is a safe bet for our teams. Presto, the Facebook project from which Trino originates, was first featured in the Radar in November 2015.

32. Wiz

Trial

Wiz is another contender in the maturing cloud security platform landscape that allows its users to prevent, detect and respond to security risks and threats in one platform. Wiz can detect and alert on misconfigurations, vulnerabilities and leaked secrets both in artifacts that have yet to be deployed to live environments (container images, infrastructure code) as well as live workloads (containers, VMs and cloud services). It also contextualizes findings to the customer's specific cloud landscape to enable response teams to better understand the issue and prioritize mitigations. Our teams have had good experience with Wiz. They find that Wiz is rapidly evolving and adding new features, and they appreciate that it enables them to detect risks and threats sooner than some other similar tools as it continuously scans for changes.

33. ActivityPub

Assess

With the current upheaval in the micro-blogging platform space, the ActivityPub protocol is gaining prominence. ActivityPub is an open protocol for sharing information such as posts, publications and dates. It can be used to implement a social media platform, but the key benefit is that it delivers interoperability between different social media platforms. We expect ActivityPub will play a significant role in this space, but we're mentioning it here because we're intrigued by the possibilities beyond the obvious use cases in social media. An example is ActivityPub support for merge requests, recently proposed for GitLab.

34. Azure Container Apps

Assess

Azure Container Apps is a managed Kubernetes namespace as a service that streamlines the deployment of containerized workloads by eliminating the need for intricate maintenance of Kubernetes clusters and underlying infrastructure components, consequently diminishing operational and administrative burdens. However, it's essential to tread carefully while considering this option; currently in its developmental phase, it has exhibited inconsistencies in the Azure portal's representation of its capabilities and encounters integration hurdles, particularly with the standard Terraform provider for Azure lagging in mirroring the tool's actual functionalities. Given all this, we recommend assessing this tool carefully.

35. Azure OpenAI Service

Assess

With the huge interest in generative AI, many solutions have sprung up to access the major models. If considering or already using Azure, then it's worth assessing Azure OpenAI Service. It provides access to OpenAI's GPT-4, GPT-35-Turbo and Embeddings models through a REST API, a Python SDK and a web-based interface. The models can be adapted to tasks such as content generation, summarization, semantic search and natural language to code translation. Fine-tuning is also available via few-shot learning and the customization of hyperparameters. In comparison to OpenAI's own API, Azure OpenAI Service benefits from Azure's enterprise-grade security and compliance features; it's also available in more regions, although availability is limited for each of the larger geographic regions, and, as of writing, India is not included.

36. ChatGLM

Assess

There are many emerging large language models (LLMs) in the English-speaking world. Although these models are usually pretrained with multiple languages, their performance in other languages may not be as good as in English. ChatGLM, developed by Tsinghua University, is an open bilingual language model optimized for Chinese conversation based on the General Language Model architecture. Since Chinese can be more complex than English with its different word segmentation and grammar, it's important to have an LLM optimized for Chinese. Our team found ChatGLM beat other LLMs in accuracy and robustness when we built a Chinese emotion detection application for a call center. Considering many LLMs aren't available in China due to licensing or regional restrictions, ChatGLM became one of the few open-source options.

37. Chroma

Assess

Chroma is an open-source vector store and embedding database, useful for enhancing applications powered by large language models (LLMs) by facilitating the storage and utilization of domain knowledge in LLMs, which typically lack internal memory. Particularly in text-to-text applications, Chroma can automate the intricate process of generating word embeddings and analyzing similarities between them and query embeddings, thereby considerably streamlining operations. It also gives you the option to store custom embeddings, fostering a blend of automation and customization. In light of its capabilities to enhance the functionality of LLM-powered applications, we advise teams to assess Chroma, tapping into its potential to refine the way domain knowledge is integrated into such applications.

38. Kraftful

Assess

UX research platforms such as Dovetail offer organizations a tool to understand and improve their customer experience. With it, businesses are able to quickly and easily gain valuable insights into their customer's needs, preferences and behaviors by collecting and analyzing data from customer feedback, surveys, interviews and more. Sentiment analysis, customer segmentation, market research or data analysis and insight generation are valuable tasks in product development — these match what LLMs are good at, hence we see a great potential for disruption in the product development field.

Kraftful — a self-described copilot for product builders — has taken the lead. It's only in beta and you must provide your email to access the feature list. We've played with it and seen great results. You can plug more than 30 sources of user feedback into the platform and it will analyze the data and identify feature requests, common complaints, what users love about the product and even name your competitors. To gather more details, you can ask questions like you would to ChatGPT or Google Bard — the benefit here is it's optimized for your data. Once you prioritize what will be addressed from the user's feedback, Kraftful generates user stories for you based on all underlying data — including acceptance criteria — making it a great assistant to even very experienced product managers and business analysts.

39. pgvector

Assess

With the rise of Generative AI-powered applications, we see a pattern of storing and efficiently searching embeddings vectors for similarities. pgvector is an open-source vector similarity search extension for PostgreSQL. We quite like it because it enables us to search the embeddings in PostgreSQL without moving the data to another store just for similarity search. Although there are several specialized vector search engines, we want you to assess pgvector.

40. Pinecone

Assess

Pinecone is a fully managed, developer-friendly and cloud-native vector database with a simple API and no infrastructure hassles. Pinecone serves filtered query results with low latency at the scale of billions of vectors. Our teams have found vendor databases and Pinecone in particular very helpful and quick to get started for use cases like storing a team's knowledge base or help desk portal content rather than fine-tuning complex LLMs.

41. wazero

Assess

wazero is a zero dependency WebAssembly (WASM) run time written in Go. Although the run time itself is language neutral, we wanted to highlight wazero for Go developers because it offers a convenient way to extend your Go programs with WASM modules written in any conformant languages. It has no dependency on CGO, so you can easily cross compile your Go applications to other platforms. Although you have a choice when it comes to run times for WASM, we think wazero is worth assessing.

Tools

Adopt

42. dbt
43. Mermaid
44. Ruff
45. Snyk

Trial

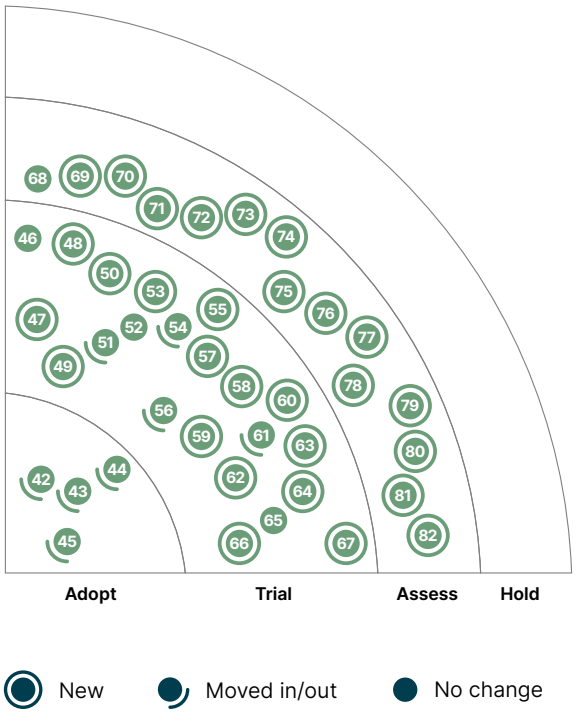
46. AWS Control Tower
47. Bloc
48. cdk-nag
49. Checkov
50. Chromatic
51. Cilium
52. Cloud Carbon Footprint
53. Container Structure Tests
54. Devbox
55. DX DevEx 360
56. GitHub Copilot
57. Insomnia
58. IntelliJ HTTP Client plugin
59. KEDA
60. Kubeconform
61. mob
62. MobSF
63. Mocks Server
64. Prisma runtime defense
65. Terratest
66. Thanos
67. Yalc

Assess

68. ChatGPT
69. Codeium
70. GitHub merge queue
71. Google Bard
72. Google Cloud Workstations
73. Gradio
74. KWOK
75. Llama 2
76. Maestro
77. Open-source LLMs for coding
78. OpenCost
79. OpenRewrite
80. OrbStack
81. Pixie
82. Tabnine

Hold

—



42. dbt

Adopt

dbt continues to be our tool of choice for data transformations in the ELT workflow. We like that it lends itself to engineering rigor and enables practices like modularity, testability and reusability of SQL-based transformations. dbt is available both as an open-source and commercial SaaS product and has a healthy ecosystem, including a community hub with packages for unit testing, data quality and data observability, to name a few. Packages worth highlighting include dbt-expectations and dbt-unit-testing which facilitate data quality checks and unit testing of the transformations, respectively. dbt integrates well with a variety of cloud data warehouses, lakehouses and databases, including Snowflake, BigQuery, Redshift, Databricks and Postgres. When working with structured data where one can set up transformations as SQL, our teams prefer dbt — which is why we're moving it to Adopt.

43. Mermaid

Adopt

Mermaid lets you generate diagrams from a Markdown-like markup language. Since we last featured it in the Radar, Mermaid has added support for many more diagrams and integrations with source code repositories, IDEs and knowledge management tools. Notably, it's supported natively in popular source code repositories such as GitHub and GitLab, enabling the embedding of and easy updates to Mermaid diagrams in the middle of Markdown documentation. Many of our teams gravitate toward Mermaid as their diagram-as-code tool due to its ease of use, multitude of integrations and wide variety of supported diagram types that keep growing.

44. Ruff

Adopt

Ruff is a relatively new linter for Python. When it comes to linters, for us the question is not whether to use a linter but which linter to use. Ruff stands out for its out-of-box experience and its speed. It has more than 500 built-in rules and readily replaces Flake8, including many of that linter's plugins. The Ruff team's claims about its performance are borne out by our experience; it really is at least an order of magnitude faster than other linters which is a huge benefit, because it helps reduce build times on large codebases. For these reasons, Ruff has become our default choice as a Python linter.

45. Snyk

Adopt

Snyk provides both static application security testing (SAST) and software component analysis (SCA) tests to help you find, fix and monitor security issues throughout the software development lifecycle. Its broad range of features is designed to speed up the feedback loop, favoring a shift-left approach instead of the security sandwich anti-pattern. As one of the best security platforms available today, Snyk stands out because of its ability to identify a wider range of issues, enabled mainly by a dedicated research team adding to its vulnerability database. But there's room for improvement: the dashboard currently doesn't provide an easy way to filter noise down to specific actionable information; depending on the language ecosystem, SCA-based integrations can output false positives compared to pipeline-based integrations because Snyk has to guess what the resolved dependencies are; automated resolution is not consistently successful; and significant integration investment is required in order to achieve proper gatekeeping or to establish an SBOM in high regulatory environments. Despite these shortcomings, many of our enterprise clients have adopted Snyk; we too are using it for our IT function.

46. AWS Control Tower

Trial

Multi-team account management is a challenge in AWS, especially in setup and governance. AWS Control Tower addresses this challenge by simplifying setup and automating governance; it addresses regulatory requirements with guardrails. AWS Control Tower has a built-in Account Factory that helps automate the account provisioning workflow. Among other things, you can update, unmanage and close accounts that you create and provision through Account Factory. Due to its lack of automation and customization, Amazon introduced AWS Control Tower Account Factory for Terraform (AFT). AFT allows you to provision customizations to send webhooks or take specific actions that allow for the integration with other tools to kick off jobs as part of the account creation process. One of the use cases leveraged by our team was to manage a set of out-of-the box items for accounts that were set-and-forget configurations for baselining and creating access for roles for GitHub Actions. This resulted in providing developers with an account that's security baselined with a fully integrated VPC, ready to receive workload via GitHub Actions. Our teams have reported great results using AWS Control Tower to manage accounts, as a single access control for multiple teams, and with leveraging AFT in their workloads.

47. Bloc

Trial

Bloc is a reactive state management library for Flutter. Among the available state management options for Flutter, we want to highlight Bloc because our teams have had good experience with the library when building complex mobile applications. Structurally organizing code around the BLoC pattern resulted in clean separation of business logic from the presentation layer as the UI Widgets communicate via streams and event sinks with business logic. Bloc also has good plugin support in both IntelliJ and VSCode IDEs.

48. cdk-nag

Trial

cdk-nag identifies and reports security and compliance issues in AWS CDK applications or CloudFormation templates. It comes with several so-called packs of rules: a general AWS pack that includes checks for what AWS considers best practices, as well as packs for HIPAA, NIST, and PCI compliance. You can add additional rules as needed. Rules can result in either warnings or errors, both of which are included in reports generated by the tool. When errors are present, the cdk deploy command will not do deployments. If the cause of the error can't be fixed in time, you can still deploy with the error present but suppressed. Obviously, this should only be done in exceptional cases.

49. Checkov

Trial

Checkov is a specialized static security scanner for infrastructure as code (IaC). It supports a broad array of infra languages, including Kubernetes manifests, Helm charts, CloudFormation templates and Terraform. Easily deployable in CI/CD pipelines, it safeguards against potential security gaps in diverse cloud infrastructure configurations. Leveraging a set of default rules, it identifies common security scenarios with detailed remediation advice available on its website. Checkov supports custom rules and uses YAML for simple guideline definitions or Python for crafting more complex ones. Our teams have successfully used Checkov to enhance security during infrastructure deployments, appreciating the early warnings it provides on potential issues before deployment.

50. Chromatic

Trial

Chromatic is a visual regression testing tool to help catch UI regression in web applications. It works by taking snapshots of UI components and comparing them against previous snapshots when they change. It's a hosted service that integrates with popular cloud code hosting services. Built on top of Storybook, it does component-level visual regression testing. It can render the components in different viewport widths for responsive testing and integrates with CI workflows, generating the UI changeset for every commit, which makes it easy to review. Our teams find the visual difference between Chromatic and other tools in this space much better; the ability to highlight changes visually makes it particularly useful.

51. Cilium

Trial

eBPF is famous for its application transparency, high performance and low overhead. Thus the cloud-native community has been exploring its use case for service mesh without sidecar. Cilium is an open-source project that provides networking, security and observability for cloud-native environments such as Kubernetes clusters and other container orchestration platforms. It provides a simple flat Layer 3 network to routing or overlay and is L7 protocol aware. By decoupling security from addressing, Cilium could play a significant role as a new network protection layer. We've seen the adoption of Cilium among some cloud providers and have also used it in Thoughtworks projects. The community is still discussing whether eBPF can replace sidecar, but there appears to be consensus that some mesh features cannot or should not be executed in the kernel. In addition, applying Cilium also requires eBPF-related experience. Based on the positive results in our project, we recommend you try this technology yourself.

52. Cloud Carbon Footprint

Trial

Cloud Carbon Footprint (CCF) is an open-source tool that estimates carbon emissions for cloud workloads across the major cloud service providers. It queries cloud APIs for resource usage data and uses multiple sources to track carbon emissions. Following a published methodology, CCF combines these into emission estimates and provides a visualization of the data over time. Cloud providers have started adding similar offerings to their platforms, but organizations are still deploying CCF because it has all of the following features: It's open-source, designed to be extended, works across multiple clouds and has a transparent, published methodology. In addition, it also includes estimates for scope 2 and scope 3 emissions — for electricity use and hardware production, respectively. In our experiments, the estimates between different tools have varied, which is not a huge surprise given that all tools in this space make estimates and multiply estimated numbers. However, settling on one tool, taking a baseline and improving from that baseline is the key usage scenario we've come across, and tools like Kepler may reduce the need for estimates in the future. CCF also delivers GCP and AWS-sourced optimization recommendations, which not only help reduce your cloud carbon footprint but can also become part of a wider cloud cost optimization strategy. Thoughtworks is a significant contributor to CCF.

53. Container Structure Tests

Trial

Container Structure Tests (CST) is a tool developed by Google to test the structure of a container image. CST can be used to check the existence or absence of a certain file in the image's file system, to verify the content of a file, to check the output or errors inside a specific command issued in the container and to check the metadata of the container image (i.e., labels, entrypoint and command) which helps ensure compliance with the CIS Docker Benchmark. We've had good experiences with CST and recommend you give it a try. In addition to preventing vulnerabilities — checking whether the container is exposing unnecessary ports — we also used it to validate that each Docker container passes all requirements necessary for it to be deployed and to run an application in the enterprise's platform. One of these requirements was having an observability agent installed in the image. It's important to be aware that CST isn't officially supported by Google, which could impact maintenance.

54. Devbox

Trial

Devbox is a terminal-based tool that provides an approachable interface for creating reproducible, per-project development environments, leveraging the Nix package manager without using virtual machines or containers. Our teams use it to eliminate version and configuration mismatches of CLI tools and custom scripts in their per-project development environments, on top of the standardization that per-language package managers provide. They found that it notably streamlines their onboarding workflow because once it has been configured for a codebase, it takes one CLI command (`devbox shell`) to stand it up in a new machine. Devbox supports shell hooks, custom scripts and devcontainer.json generation for integration with VSCode.

55. DX DevEx 360

Trial

DX DevEx 360 is a survey-based tool that helps find leading signals of developer productivity by focusing on frictions that developers face in their day-to-day work, such as the code review process, code quality, ability to do deep work and more. The survey is designed by Nicole Forsgren and Margaret-Anne Storey, who have led previous efforts like DORA and SPACE, among other experts.

Our platform engineering teams have used DX DevEx 360 successfully to understand developer sentiment and identify friction points to inform the platform roadmap. Unlike similar tools, with DX DevEx 360 we've received a response rate of 90% or above, often with detailed comments from developers on problems and ideas for improvements. We also appreciate that the tool makes results transparent to engineers in the company instead of just managers and that it enables team-by-team breakdown to enable continuous improvement for each team's context.

56. GitHub Copilot

Trial

GitHub Copilot is used by many of our teams to help them write code faster. Overall, most of our developers find it very useful and would be cross if we took it away from them. We've been collating and sharing many of our experiences with Copilot through a series on Generative AI and a guide on getting started with Copilot. Note that GitHub Copilot can be used with any codebase, not just codebases hosted on GitHub.

We're also excited that Copilot's chat feature from the Copilot X roadmap has become more widely available since we last featured it in the Radar. It is a powerful addition to Copilot's in-line assistance feature. The availability of a chat interface inside the IDE improves the discoverability of commonly searched information, and integration with the open editor context makes it easy to explore errors or ask the chat to assist with tasks related to the code in focus.

57. Insomnia

Trial

Since Postman announced in May 2023 that it would be sunsetting Scratch Pad mode with its offline capabilities, teams that need to keep API workspace data off third-party servers have had to find alternatives. Insomnia is one such alternative: it's a free and open-source desktop application designed for API testing, development and debugging. Although Insomnia supports online syncing, it'll let you keep API workspace data offline. Our teams have found migration from Postman for manual API testing seamless as its features are similar, and it allows importing of Postman collections. Despite our teams' positive experiences with Insomnia, we're keeping an eye on other developing alternatives of various forms — from GUI tools like Insomnia that are drop-in alternatives, to CLI tools like HTTPie, to IDE plugins such as IntelliJ HTTP client plugin.

58. IntelliJ HTTP Client plugin

Trial

IntelliJ HTTP Client plugin allows developers to create, edit and execute HTTP requests inside the code editor, simplifying the development process when building and consuming APIs. It's becoming increasingly popular among our teams, who appreciate its user-friendly features and convenience. Its standout features include support for private files which safeguards sensitive keys by excluding them from git by default, version control and the ability to utilize variables which enhances the developer experience. Given its capacity to streamline developer workflows and bolster security measures, we recommend trying this tool.

59. KEDA

Trial

KEDA, the Kubernetes Event-Driven Autoscaler, does exactly what its name suggests: it enables the scaling of a Kubernetes cluster in relation to the number of events that must be processed. In our experience, using leading indicators like queue depth — rather than trailing indicators like CPU usage — is preferable. KEDA supports different event sources and comes with a catalog of more than 50 scalers for various cloud platforms, databases, messaging systems, telemetry systems, CI/CD systems and more. Our teams report that the ease with which KEDA can be integrated has allowed them to keep functionality in microservices in Kubernetes where they otherwise might have considered porting some of the event handling code to serverless functions.

60. Kubeconform

Trial

Kubeconform is a streamlined tool for validating Kubernetes manifests and custom resource definitions (CRD). Easily deployable in CI/CD pipelines or local machine setups, it fosters confidence by validating resources prior to deployment, thereby mitigating potential errors. Given its track record in enhancing operational assurance, especially with templated resources shared across teams, we recommend trialing Kubeconform to bolster the security and efficiency of your resource validation processes.

61. mob

Trial

mob is a command-line tool for seamless git handover in remote pair or mob programming. It hides all the version control paraphernalia behind a command-line interface that makes participating in mob programming sessions simpler. It also provides specific advice around how to participate remotely, for example, to "steal the screenshare" in Zoom rather than ending a screenshare, ensuring the video layout doesn't change for participants. Several of our teams highly recommend mob, and it has become an integral part of our toolchain in remote pair or mob programming.

62. MobSF

Trial

MobSF is an open-source, automated static and dynamic security testing tool for detecting security vulnerabilities in iOS and Android mobile apps. It scans both app sources and binaries and provides detailed reports about vulnerabilities. MobSF is distributed as Docker images and comes with easy-to-use REST APIs and, via mobsfscan, can be integrated into CI/CD pipelines. Our experience using MobSF for security testing Android apps has been positive; we recommend trying it for your mobile app security testing needs.

63. Mocks Server

Trial

Mocks Server is a Node.js-based API mocking tool valued by our teams for its ability to replicate intricate API responses, headers and status codes. The dynamic response generation supports the simulation of diverse scenarios, which allows for the rigorous testing of API interactions. Mocks can be described as YAML or JSON and managed through the CLI, REST API or JavaScript code. Mocks Server's features include request matching, proxying and record-playback features, which facilitate the emulation of realistic API interactions. We particularly like the integration with Docker containers, which makes it easy to deploy the server consistently between environments so it can be versioned and maintained as another artifact of the ecosystem. Its straightforward approach aligns with our emphasis on simplicity and efficiency in development processes. We look forward to using Mocks Server more extensively as our testing strategy evolves along with our solutions.

64. Prisma runtime defense

Trial

Prisma runtime defense, which is a part of the Prisma Cloud suite, offers a new approach to container security. It employs a mechanism to build a model of a container's expected behavior, and then detects and blocks anomalous activities when some variance is found during the run time. It monitors container processes, network activities and file systems for patterns and changes that indicate an attack might be underway and blocks according to the configured rules. The models that learn what constitutes "normal" behaviors are built from both the static analysis of docker images and dynamic behavioral analysis for a preconfigured period. Our teams have found the results from our usage promising.

65. Terratest

Trial

Terratest continues to be an interesting option for infrastructure testing. It is a Golang library that makes it easier to write automated tests. Using infrastructure-as-code tools such as Terraform, you can create real infrastructure components (such as servers, firewalls or load balancers) to deploy applications on them and then validate the expected behavior using Terratest. At the end of the test, Terratest can undeploy the apps and clean up resources. Our teams report that this approach of testing deployed infrastructure components fosters confidence in the infrastructure as code. We see our teams writing a variety of infra security tests for application components and their integrations. These include detecting misconfigurations, verifying access control (e.g., to check whether certain IAM roles or permissions are correctly configured or to ensure only authorized users have access to specific resources) and network security tests to verify prevention of unauthorized traffic to sensitive resources to name a few. This allows security testing to be shifted left and provides feedback during development itself.

66. Thanos

Trial

Although Prometheus continues to be a solid option for a self-managed observability stack, many teams managing modern cloud-native distributed systems bump into its single-node limitations as their metrics grow in cardinality and sheer volume, and when they start needing high-availability setup. Thanos extends Prometheus by adding features that make it suitable for large-scale, long-term and highly available monitoring. It does this, for example, by introducing components that will read data from Prometheus instances and store it in object stores, manage retention and compaction in the object stores and federate querying across multiple Prometheus instances. Our teams have found the migration from Prometheus seamless, as Thanos maintains compatibility with the Prometheus query API. This means they can continue to use their existing dashboards, alerting tools and other tools that integrate with the Prometheus API. Although our teams have been successful with Thanos, we also recommend keeping an eye on Cortex, as another way to extend Prometheus.

67. Yalc

Trial

Yalc is a simple local JavaScript package repository and a tool for publishing and using packages across a local development environment. It's a more reliable alternative to the npm link command, which has some constraints. Yalc is useful when working with multiple packages, especially when some use yarn and some use npm. It's also helpful for testing the packages locally before publishing them to a remote registry. In our experience, Yalc is valuable in a multi-package setup and speeds up front-end and other JavaScript app development workflows.

68. ChatGPT

Assess

ChatGPT continues to attract attention. Imaginative use cases and innovative approaches to prompting mean it's gaining expanding utility over time. GPT4, the large language model (LLM) that powers ChatGPT, now also has the ability to integrate with external tools such as a knowledge management repository, sandboxed coding environment or web search. The recent introduction of ChatGPT Enterprise may help ease intellectual property concerns, while providing "enterprise" features such as usage tracking and better user management through SSO.

Although ChatGPT's ability to "write" code has been much vaunted, we think organizations should be looking at using it across the full software lifecycle to improve efficiency and reduce errors. For example, ChatGPT can provide additional perspectives or suggestions for tasks as diverse as requirements analysis, architectural design or the reverse engineering of legacy systems. We still think ChatGPT is best used as an input to a process — such as helping with a first draft of a story or the boilerplate for a coding task — rather than a tool that produces "fully baked" results. That said, its capabilities are improving each week, and some programming tasks may now be fully possible by careful prompting, which is an art in itself.

69. Codeium

Assess

In the AI-powered coding assistants space, Codeium is one of the more promising products. Similar to Tabnine, Codeium tries to address some of the biggest concerns companies have about coding assistants: They reduce at least some of the open-source licensing concerns, because they do not train their models with code from repositories with nonpermissive licenses. They also offer self-hosting for the tool so you don't have to send your code snippets to a third-party service. Codeium stands out for its broad support of IDEs and notebook services, and although it hasn't been around for as long as GitHub Copilot or Tabnine, our first impressions of the product have been positive.

70. GitHub merge queue

Assess

We've long been advocates of short-lived developer branches that are merged frequently into the main code branch, which is kept ready for deployment. This practice of trunk-based development goes hand-in-hand with continuous integration and, when conditions permit, results in the fastest feedback cycles and most efficient developer flow. However, not everyone favors this approach, and we frequently adapt our style to accommodate our clients' practices. Sometimes this includes long-lived feature branches followed by pull requests that must be manually reviewed and approved before they're merged into the main branch. In these situations, we use the new GitHub merge queue feature.

It allows us to automatically queue up incoming pull requests and merge them into a special branch in the order they were received. We then have the option of automating our own "merge checks" to prevent incompatible commits. This essentially simulates trunk-based development (even though PRs have not been merged into the main code branch yet) and allows developers to test their features in context without having to wait for the pull request to be approved. With GitHub merge queue, you get the benefits of trunk-based development even when you're not able to fully commit to it.

71. Google Bard

Assess

Google Bard is a generative AI chatbot developed by Google AI. Much like ChatGPT, it's conversational and able to communicate and generate human-like text in response to a wide range of prompts and questions. Bard is powered by Google's Pathways Language Model (PaLM 2), which is a large language model (LLM) that's trained on a massive data set of text and code. Bard is able to generate text, translate languages, write different kinds of creative content and answer your questions in an informative way.

Bard can also be used as a guidance tool for software development. Sometimes, our developers have found it useful to get some coding suggestions or best practices and infrastructure configurations for different scenarios. We also experimented with Bard for the Radar's language translations and got decent results to create the first draft of the text. Although the tool is still in development, which is why it can be a bit slower when compared to ChatGPT, we encourage developers to explore the tool and assess its potential benefits.

72. Google Cloud Workstations

Assess

Google Cloud Workstations is GCP's Cloud Development Environment (CDE) offering. It offers fully managed containerized development environments that are accessible through SSH, HTTPS, VSCode and Jetbrains IDEs among others, giving developers the illusion of connecting to a local environment. Google Cloud Workstations allows administrators to make the containerized development environments part of a private network and to be either publicly or privately accessible. This ability to tweak networking configurations, along with support to create the environments with either custom or predefined images, makes Google Cloud Workstations, in our view, worth assessing for organizations looking for a secure CDE solution within their own GCP perimeter. If you're considering Google Cloud Workstations, we recommend that you test your networking configuration before rolling it out widely, as high latency can become a real friction to the developer experience of these containers.

73. Gradio

Assess

Gradio is an open-source Python library that allows for the quick-and-easy creation of interactive web-based interfaces for ML models. A graphical user interface on top of ML models enables a better understanding of the inputs, constraints and outputs by nontechnical audiences. Gradio supports many input and output types — from text and images to voice — and has emerged as a go-to tool for rapid prototyping and model evaluation. Gradio lets you easily host your demos on Hugging Face or run them locally and enable others to access your demo remotely with a "XXXXX.gradio.app" url. For example, the famous DALL-E mini experiment leverages Gradio and is hosted on Hugging Face Spaces. Our teams have been happy using this library for experimentation and prototyping, which is why we put it in Assess.

74. KWOK

Assess

KWOK (Kubernetes WithOut Kubelet) is a tool for simulating the lifecycle of fake nodes and pods in order to test the control plane of a Kubernetes cluster. It is difficult to stress test custom Kubernetes controllers and operators without a significantly large cluster. However, with KWOK you can easily set up a cluster with thousands of nodes on your laptop without it consuming a significant amount of CPU or memory. This simulation enables different configurations of node types and pods to test various scenarios and edge cases. If you need a real cluster to test your operators and custom resource definitions (CRDs), we recommend kind or k3s; but if you only need to simulate a large number of fake nodes, then we encourage you to assess KWOK.

75. Llama 2

Assess

Llama 2, from Meta, is a powerful language model that is free for both research and commercial use. It's available both as a raw pretrained model and, fine-tuned, as Llama-2-chat for conversation and Code Llama for code completion. Since it's available in a variety of sizes — 7B, 13B, and 70B — Llama 2 is a good option for a self-hosted LLM, if you want to control your data.

Meta describes Llama 2 as "open source," a claim that has attracted some criticism. Meta's license and acceptable use policy put restrictions on commercial use for some users and also restricts the use of the model and software for certain purposes. The Llama 2 training data is not open, which can hamper the ability to understand and change the model. That said, the availability of a powerful, capable model in at least a "semi-open" form is welcome.

76. Maestro

Assess

Maestro is a new cross-platform mobile UI test automation tool with built-in tolerance for flakiness and variance in application load times because of network or external factors. With a declarative YAML syntax, it makes it easy to write and maintain automated tests for mobile apps. It supports iOS and Android native apps, React Native and Flutter apps, as well as a variety of features for automating complex mobile UI interactions, such as tapping, scrolling and swiping. Maestro is distributed as a single binary for ease of use, runs in interpreted mode and makes it easy to author new tests thanks to features like continuous mode. Maestro still lacks specific features like support for iOS devices, but the tool is rapidly evolving.

77. Open-source LLMs for coding

Assess

GitHub Copilot is a valuable tool for coding assistance while developing software. Under the hood, LLMs can power seamless developer experiences through inline code assistance, code fine-tuning, conversational support in the IDE and much more. Most of these models are proprietary and can only be used via subscription services. The good news is you can use several open-source LLMs for coding. If you're in a space where you need to build your own coding assistance service (such as a highly regulated industry), look at models like StarCoder and WizardCoder. StarCoder is trained with a large data set maintained by BigCode, and Wizardcoder is an Evol-Instruct tuned StarCoder model.

We've used StarCoder in our experiments and found it to be useful for generating structured software engineering elements such as code, YAML, SQL and JSON. Based on our experiments, we found both the models to be receptive to in-context learning using few-shot examples in the prompt. Nonetheless, for specific downstream tasks (such as SQL generation for a specific database like Postgres) the models needed fine-tuning. Recently, Meta unveiled its Code Llama, a code-specialized version of Llama 2. Be sure to proceed with caution when using these open-source models. Consider their license, the license of the code and of the data set used to train the model. Carefully assess these aspects before you choose any of these coding LLMs for your organization.

78. OpenCost

Assess

OpenCost is an open-source project for monitoring infrastructure cost that surfaces cost at the granularity of Kubernetes objects (pods, containers, clusters, etc.), covering various in-cluster resources (CPU, GPU, RAM, storage, network). It has integrations with multiple cloud provider APIs to obtain billing data and can be configured with pricing for on-premise Kubernetes clusters. OpenCost is the cost allocation engine originally built and still used by Kubecost, but it can also be used on its own. Cost allocation data from OpenCost can be exported into CSV files or to Prometheus for further analysis and visualization. Our teams are keenly watching the developments of tools like OpenCost and Kubecost that enable cost visibility for product and platform teams in organizations that have adopted Kubernetes. In these early stages, they find that OpenCost doesn't work well yet with certain workloads such as short-lived spot instances often used in data pipelines.

79. OpenRewrite

Assess

We've seen several use cases for code intelligence tools: moving to a new API version of a widely used library, understanding the impact of a just discovered vulnerability in such a library across an enterprise or applying updates to many services that were created from the same template. Sourcegraph is still a popular tool in this space, and OpenRewrite is another tool we want to highlight. While our teams have mostly used it in Java for narrow problems, like updating services created through a starter kit, it continues to broaden its coverage of languages and use cases. We like that it comes bundled with a catalog of recipes, which describe the changes to be made, for example for migrating commonly used frameworks across versions. The refactoring engine, bundled recipes and build tool plugins are open-source software, which makes it easier for teams to reach for OpenRewrite just when they need it. It remains to be seen how the maturing space of code intelligence tools, which are all based on parsing the source code into an abstract syntax tree (AST), will be impacted by the rapid developments in the space of LLMs.

80. OrbStack

Assess

OrbStack is a way to run Docker containers on macOS; our developers have found it to be more lightweight, faster and simpler to set up and use than Docker Desktop and Colima. The tool is still under development and therefore has fewer features, but it's already showing some great potential with its simplicity and speed. You can also use OrbStack to create and manage Linux VMs on macOS.

81. Pixie

Assess

Pixie is an observability tool for Kubernetes native applications. It takes an interesting approach toward observability by leveraging eBPF to automatically collect telemetry data from multiple data sources. Collected telemetry data is stored locally in each node and processed centrally via its control plane API. Overall, we find Pixie worthwhile to assess for observability in the Kubernetes ecosystem.

82. Tabnine

Assess

Tabnine is a contender in the currently busy space of coding assistants. It provides in-line code completion suggestions and a chat directly in the IDE. Similar to GitHub Copilot, Tabnine has been around since before the current hype cycle and is therefore one of the more mature products in the space. Unlike Copilot, it uses a model that is only trained on permissively licensed code and offers a version to self-host for organizations that worry about sending their code snippets to a third-party service. Tabnine is available as a limited free version as well as a paid version, which has more comprehensive suggestions and also offers a mode with a local (albeit less powerful) model that you can use without internet connection.

Languages and Frameworks

Adopt

83. Playwright

Trial

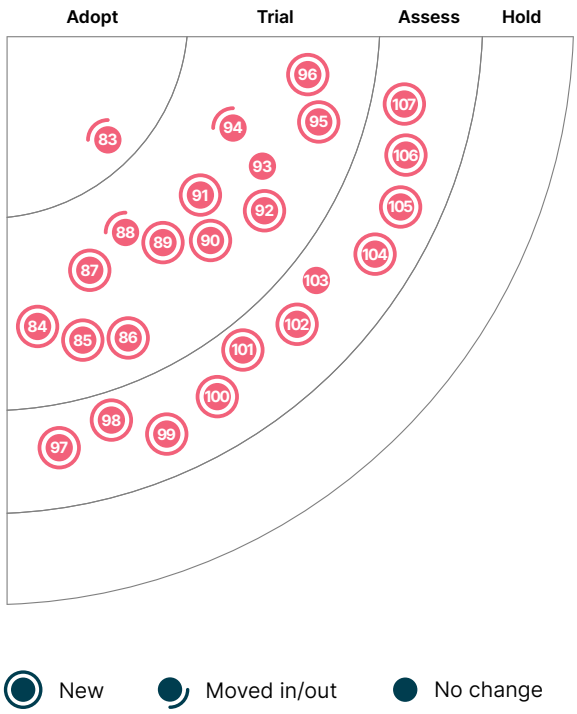
84. .NET Minimal API
85. Ajv
86. Armeria
87. AWS SAM
88. Dart
89. fast-check
90. Kotlin with Spring
91. Mockery
92. Netflix DGS
93. OpenTelemetry
94. Polars
95. Pushpin
96. Snowpark

Assess

97. Baseline Profiles
98. GGML
99. GPTCache
100. Grammatical Inflection API
101. htmx
102. Kotlin Kover
103. LangChain
104. LlamaIndex
105. promptfoo
106. Semantic Kernel
107. Spring Modulith

Hold

—



83. Playwright

Adopt

With Playwright you can write end-to-end tests that run in Chrome, Firefox and WebKit. By using the Chrome DevTools Protocol (CDP) Playwright can offer new features and eliminate many of the issues seen with WebDriver. Using the same API eliminates many of the issues that WebDriver has and introduces a set of new features. Chromium-based browsers implement CDP directly. To support Firefox and Webkit, though, the Playwright team has to submit patches to these browsers, which may sometimes limit the framework.

Playwright's many features include: Built-in auto-waits, which result in tests that are more reliable and easier to understand; browser contexts, which let you test that persisting sessions across tabs work properly; and the ability to simulate notifications, geolocation and dark mode settings. Our teams are impressed with the stability Playwright brings to the test suite and like that they get feedback more quickly by running tests in parallel. Other features that set Playwright apart include better support for lazy loading and tracing. Although Playwright has some limitations — component support is currently experimental, for example — our teams consider it the go-to test framework and in some cases are migrating away from Cypress and Puppeteer.

84. .NET Minimal API

Trial

ASP.NET core MVC has proven to be a powerful and flexible approach for building web applications that host APIs. However, its flexibility brings a certain amount of complexity with it, including boilerplate and conventions that aren't always obvious. The routing provided by ASP.NET allows multiple services to be hosted in a single application, but in today's world of serverless functions and independently deployable microservices, that flexibility might be overkill. .NET Minimal APIs provide a simple approach to implementing a single-API web application in the .NET ecosystem. The Minimal API framework can implement an API endpoint with just a few lines of code. Minimal API joins the new generation of API frameworks — including Micronaut, Quarkus and Helidon — which are optimized for lightweight deployments and fast startup times. We're interested in the combination of Minimal APIs and .NET 7 Native AOT for implementing simple, lightweight microservices in serverless functions.

85. Ajv

Trial

Ajv is a popular JavaScript library used to validate a data object against a structure defined using a JSON Schema. Ajv is fast and flexible for validating complex data types. It supports a wide range of schema features, including custom keywords and formats. It's used by many open-source JavaScript applications and libraries. Our teams have used Ajv for implementing consumer-driven contract testing in CI workflows, and, along with other tools for generating mock data with the JSON Schema, it's very powerful. In the TypeScript world, Zod is a popular alternative that has a declarative API for defining the schema and validating the data.

86. Armeria

Trial

Armeria is an open-source framework for building microservices. Our teams have used it to build asynchronous APIs, and we quite like the approach to address cross-cutting concerns, like distributed tracing or circuit breakers, via service decorators. The framework includes port reuse for both gRPC and REST traffic among other clever design choices. With Armeria, we can incrementally add new features in gRPC on top of an existing codebase in REST or vice versa. Overall, we find Armeria to be a flexible microservices framework with several out-of-the-box integrations.

87. AWS SAM

Trial

The AWS Serverless Application Model (SAM) is an open-source framework for building serverless applications on the AWS Cloud infrastructure. Previously we blipped Serverless Framework as a popular framework for deploying serverless services on various cloud providers, primarily AWS Lambda-based services. AWS SAM has gained popularity in recent times as the framework has come a long way from its early days. Our teams have found it very easy to set up, and they also use it for testing and debugging AWS Lambda-based services, including local executions of Lambdas for development.

88. Dart

Trial

Dart is a programming language developed by Google that supports building apps targeting multiple platforms, including web browsers, WebAssembly, desktop and mobile apps. Its adoption has been driven by the dominance of Flutter — a popular, multi-platform UI toolkit powered by Dart — in the cross-platform native mobile app framework space. In response to community feedback, Dart has evolved since its initial versions and has added built-in sound null safety in version three, in addition to a robust type system. Furthermore, Dart's ecosystem is growing rapidly, with a vibrant community and a wide range of available libraries and tools, making it attractive for developers.

89. fast-check

Trial

fast-check is a property-based testing tool for JavaScript and TypeScript, capable of automatically generating test data so a wide range of inputs can be explored without creating separate tests. This makes it easier to uncover edge scenarios. Our teams are reporting good results using fast-check in back-end testing due to its good documentation, ease of use and seamless integration with existing testing frameworks, which enhances unit testing efficiency.

90. Kotlin with Spring

Trial

Five years ago we moved Kotlin into the Adopt ring, and today many of our teams report that Kotlin is not only their default choice on the JVM but that it has displaced Java almost completely in the software they write. At the same time, microservice envy appears to be fading — we've noticed people starting to explore architectures with larger deployable units, using frameworks like Spring Modulith among others. We're aware of many good Kotlin-native frameworks and have mentioned some of them previously; however, in some cases, the maturity and feature-richness of the Spring framework is a real asset, and we've been using Kotlin with Spring successfully, usually with no or only smaller issues.

91. Mockery

Trial

Mockery is a mature Golang library that helps generate mock implementations of interfaces and simulates the behavior of external dependencies. With type-safe methods to generate call expectations and flexible ways to mock return values, it enables the tests to focus on the business logic rather than worrying about the correctness of external dependencies. Mockery uses Go generators and simplifies the generation and management of the mocks in the test suite.

92. Netflix DGS

Trial

We mostly use GraphQL for server-side resource aggregation and have implemented the server side using a variety of technologies. For services written with Spring Boot, our teams have had good experiences with Netflix DGS. It's built on top of graphql-java and provides features and abstractions in the Spring Boot programming model that make it easy to implement GraphQL endpoints and integrate with Spring features such as Spring Security. Although it's written in Kotlin, DGS works equally well with Java.

93. OpenTelemetry

Trial

We've been using OpenTelemetry as a solution for a while now and recommended trying it in previous editions. Its ability to seamlessly capture, instrument and manage telemetry data across various services and applications has improved our observability stack. OpenTelemetry's flexibility and compatibility with diverse environments have made it a valuable addition to our toolkit. We're now particularly curious about the recent release of the OpenTelemetry Protocol (OTLP) specification, which includes both gRPC and HTTP. This protocol standardizes the format and transmission of telemetry data, promoting interoperability and simplifying integrations with other monitoring and analysis tools. As we continue to explore the integration potential of the protocol, we're evaluating its long-term impact on our monitoring and observability strategy and on the general monitoring landscape.

94. Polars

Trial

Polars is an in-memory data frame library implemented in Rust. Unlike other data frames (such as pandas), Polars is multithreaded, supports lazy execution and is safe for parallel operations. The in-memory data is organized in the Apache Arrow format for efficient analytic operations and to enable interoperability with other tools. If you're familiar with pandas, you can quickly get started with Polars' Python bindings. We believe Polars, with Rust implementation and Python bindings, is a performant in-memory data frame for your analytical needs. Our teams continue to have a good experience with Polars which is why we're moving it to Trial.

95. Pushpin

Trial

Pushpin is a reverse proxy that acts as an intermediary between clients and back-end servers handling long-lived connections, such as WebSockets and Server-Sent Events. It provides a way to terminate long-lived connections from clients and means the rest of the system can be abstracted from this complexity. It effectively manages a large number of persistent connections and automatically distributes them across multiple back-end servers, optimizing performance and reliability. Our experience using this for real-time communication with mobile devices using WebSockets has been good, and we've been able to scale it horizontally for millions of devices.

96. Snowpark

Trial

Snowpark is a library for querying and processing data at scale in Snowflake. Our teams use it for writing manageable code for interacting with data residing in Snowflake — it's akin to writing Spark code but for Snowflake. Ultimately, it's an engine that translates code into SQL that Snowflake understands. You can build applications that process data in Snowflake without moving data to the system where your application code runs. One drawback: unit testing support is suboptimal; our teams compensate for that by writing other types of tests.

97. Baseline Profiles

Assess

Baseline Profiles — not to be confused with Android Baseline profiles — are Android Runtime profiles that guide ahead-of-time compilation. They're created once per release on a development machine and are shipped with the application, making them available faster than relying on Cloud Profiles, an older, related technology. The run time uses the Baseline Profile in an app or library to optimize important code paths, which improves the experience for new and existing users when the app is downloaded or updated. Creating Baseline Profiles is relatively straightforward and can lead to significant (up to 30%) performance boosts according to its documentation.

98. GGML

Assess

GGML is a C library for machine learning that allows for CPU inferencing. It defines a binary format for distributing large language models (LLMs). To do that it uses quantization, a technique that allows LLMs to run on consumer hardware with effective CPU inferencing. GGML supports a number of different quantization strategies (e.g., 4-bit, 5-bit, and 8-bit quantization), each of which offers different trade-offs between efficiency and performance. A quick way to test, run and build apps with these quantized models is a Python binding called C Transformers. This is a Python wrapper on top of GGML that takes away the boilerplate code for inferencing by providing a high level API. We've leveraged these libraries to build proof of concepts and experiments. If you're considering self-hosted LLMs, carefully assess these community-supported libraries for your organization.

99. GPTCache

Assess

GPTCache is a semantic cache library for large language models (LLMs). We see the need for a cache layer in front of LLMs for two main reasons — to improve the overall performance by reducing external API calls and to reduce the cost of operation by caching similar responses. Unlike traditional caching approaches that look for exact matches, LLM-based caching solutions require similar or related matches for the input queries. GPTCache approaches this with the help of embedding algorithms to convert the input queries into embeddings and then use a vector datastore for similarity search on these embeddings. One drawback of such a design is that you may encounter false positives during cache hits or false negatives during cache misses, which is why we recommend you carefully assess GPTCache for your LLM-based applications.

100. Grammatical Inflection API

Assess

In many languages, gender is more visible than in English, and words change based on gender. Addressing users, for example, can require words to be inflected, but it is common practice to use the masculine form by default. There is evidence that this has a negative impact on performance and attitude — and, of course, it's impolite. Workarounds using gender-neutral language can often feel clumsy. Addressing users correctly, then, is the preferred option, and with the Grammatical Inflection API, introduced in Android 14, Android developers now have an easier path to do so.

101. htmx

Assess

htmx is a small, neat HTML UI library that recently became popular seemingly out of nowhere. During our Radar discussion, we found its predecessor intercooler.js existed ten years ago. Unlike other increasingly complex pre-compiled JavaScript/TypeScript frameworks, htmx encourages the direct use of HTML attributes to access operations such as AJAX, CSS transitions, WebSockets and Server-Sent Events. There's nothing technically sophisticated about htmx, but its popularity recalls the simplicity of hypertext in the early days of the web. The project's website also features some insightful (and amusing) essays on hypermedia and web development, which suggests the team behind htmx have thought carefully about its purpose and philosophy.

102. Kotlin Kover

Assess

Kotlin Kover is a code coverage tool set designed specifically for Kotlin, supporting Kotlin JVM, Multiplatform and Android projects. The significance of code coverage lies in its ability to spotlight untested segments, which reinforces software reliability. As Kover evolves, it stands out because of its ability to produce comprehensive HTML and XML reports, coupled with unmatched precision tailored to Kotlin. For teams deeply rooted in Kotlin, we advise you to assess Kover to leverage its potential in enhancing code quality.

103. LangChain

Assess

LangChain is a framework for building applications with large language models (LLMs). To build practical LLM products, you need to combine them with user- or domain-specific data which wasn't part of the training. LangChain fills this niche with features like prompt management, chaining, agents and document loaders. The benefit of components like prompt templates and document loaders is that they can speed up your time to market. Although it's a popular choice for implementing Retrieval-Augmented Generation applications and the ReAct prompting pattern, LangChain has been criticized for being hard to use and overcomplicated. When choosing a tech stack for your LLM application, you may want to keep looking for similar frameworks — like Semantic Kernel — in this fast-evolving space.

104. LlamaIndex

Assess

LlamaIndex is a data framework designed to facilitate the integration of private or domain-specific data with large language models (LLMs). It offers tools for ingesting data from diverse sources — including APIs, databases and PDFs — and structures this data into a format that LLMs can easily consume. Through various types of "engines," LlamaIndex enables natural language interactions with this structured data, making it accessible for applications ranging from query-based retrieval to conversational interfaces. Similar to LangChain, LlamaIndex's goal is to accelerate development with LLMs, but it takes more of a data framework approach.

105. promptfoo

Assess

promptfoo enables test-driven prompt engineering. While integrating LLMs in applications, tuning of the prompts to produce optimal, consistent outputs can be time-consuming. You can use promptfoo both as a CLI and a library to systematically test prompts against predefined test cases. The test case, along with assertions, can be set up in a simple YAML config file. This config includes the prompts being tested, the model provider, the assertions and the variable values that will be substituted in the prompts. promptfoo supports many assertions, including checking for equality, JSON structure, similarity, custom functions or even using an LLM to grade the model outputs. If you're looking to automate feedback on prompt and model quality, do assess promptfoo.

106. Semantic Kernel

Assess

Semantic Kernel is the open-source version of one of the core components in Microsoft's Copilot suite of products. It's a Python library that helps you build applications on top of large language models (LLMs), similar to LangChain. Semantic Kernel's core concept is its planner, which lets you build LLM-powered agents that create a plan for a user and then execute it step by step with the help of plugins.

107. Spring Modulith

Assess

Although we were early advocates for microservices and have seen the pattern used successfully on myriad systems, we've also seen microservices misapplied and abused, often as the result of microservice envy. Rather than start a new system with a collection of separately deployed processes, it's often advisable to start with a well-factored monolith and only break out separately deployable units when the application reaches a scale where the benefits of microservices outweigh the additional complexity inherent in distributed systems. Recently we've seen a resurgence of interest in this approach and a more detailed definition of what, exactly, constitutes a well-factored monolith. Spring Modulith is a framework that helps you structure your code in a way that makes it easier to break out microservices when the time is right. It provides a way to modularize your code so that the logical concepts of domains and bounded context are aligned with the physical concepts of files and package structure. This alignment makes it easier to refactor the monolith when necessary and to test domains in isolation. Spring Modulith provides an in-process eventing mechanism that helps further decouple modules within a single application. Best of all, it integrates with ArchUnit and jmolecules to automate verification of its domain-driven design rules.

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

Thoughtworks is a global technology consultancy that integrates strategy, design and engineering to drive digital innovation. We are 11,500+ people strong across 51 offices in 18 countries. Over the last 30 years, we've delivered extraordinary impact together with our clients by helping them solve complex business problems with technology as the differentiator.

/thoughtworks

Strategy. Design. Engineering.