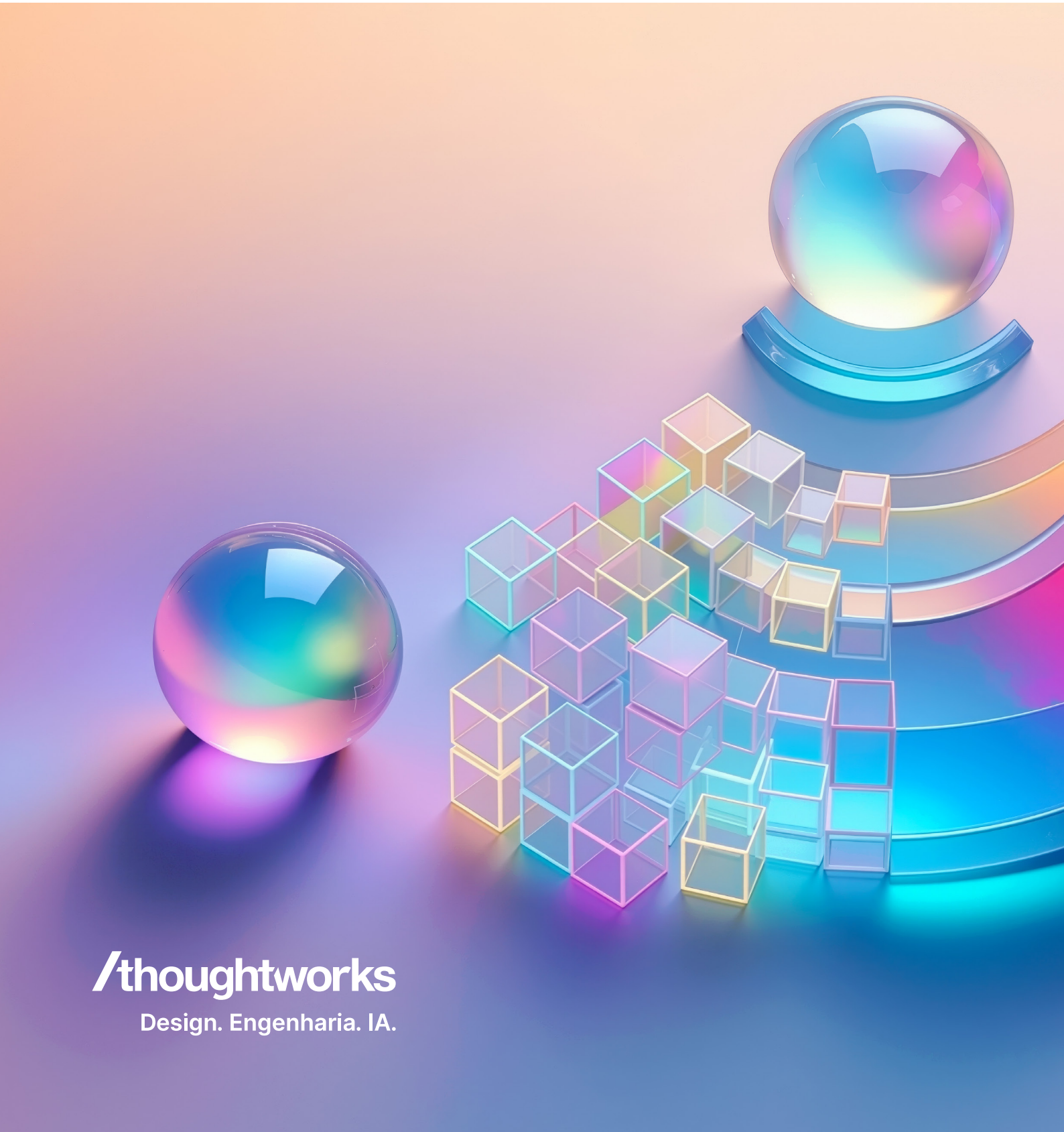
Technology Radar

Um guia de opinião sobre o universo de tecnologia atual

Volume 34
Abril 2026



/thoughtworks
Design. Engenharia. IA.

Sobre o Radar

Thoughtworkers são pessoas apaixonadas por tecnologia. Nós desenvolvemos, pesquisamos, testamos, contribuímos com código livre, escrevemos sobre e visamos a sua constante melhoria — para todas as pessoas. Nossa missão é liderar e promover a excelência de software e revolucionar a indústria de TI. Nós criamos e compartilhamos o Technology Radar da Thoughtworks para apoiar essa missão. O Conselho Consultivo de Tecnologia (Technology Advisory Board, ou TAB), um grupo de lideranças experientes em tecnologia da Thoughtworks, é responsável por criar o Radar. O grupo se reúne regularmente para discutir a estratégia global de tecnologia da empresa e as tendências tecnológicas que impactam significativamente a nossa indústria.

O Radar capta o resultado das discussões do TAB em um formato que procura oferecer valor a uma ampla gama de pessoas interessadas, de pessoas que desenvolvem software a CTOs. O conteúdo é concebido para ser um resumo conciso.

Nós encorajamos você a explorar essas tecnologias. O Radar é gráfico por natureza, agrupando os itens em técnicas, ferramentas, plataformas, linguagens e frameworks. No caso de itens que podem ser classificados em mais de um quadrante, escolhemos aquele que parece mais adequado.

Além disso, agrupamos esses itens em quatro anéis para refletir nossas opiniões atuais em relação a cada um. Para mais informações sobre o Radar, acesse: thoughtworks.com/pt/radar/faq.

Radar em um relance

A ideia por trás do Radar é rastrear tópicos interessantes, que chamamos de blips. Organizamos blips no Radar usando duas categorias: quadrantes e anéis. Os quadrantes representam as diferentes naturezas dos blips. Os anéis indicam nossa recomendação para utilizar a tecnologia.

Um blip é uma tecnologia ou técnica que desempenha um papel no desenvolvimento de software. Os blips estão "em movimento" — suas posições no Radar estão constantemente mudando — geralmente indicando que nossa confiança em recomendá-los tem crescido à medida que se movimentam entre os anéis.

Adote: Acreditamos firmemente que a indústria deveria adotar esses itens. Quando apropriado, nós os usamos em nossos projetos.

Experimente: Vale a pena ir atrás. É importante entender como desenvolver essa capacidade. As empresas devem testar a tecnologia em um projeto que possa lidar com o risco.

Avalie: Vale explorar com o objetivo de compreender como afetará sua empresa.

Cautela: Proceda com cuidado. Esses items envolvem receios significativos que exigem uma avaliação criteriosa alinhada ao seu contexto.

Novo **Mudança de anel**

Sem alterações

Nosso Radar é um olhar para o futuro. Para abrir espaço para novos itens, apagamos itens que não foram modificados recentemente, o que não é um reflexo de seus valores, mas uma solução para nossa limitação de espaço.

Contribuidoras

O Conselho Consultivo de Tecnologia (TAB) é um grupo formado por 22 tecnologistas experientes da Thoughtworks. O TAB se reúne duas vezes por ano pessoalmente e quinzenalmente por vídeochamada. Sua principal atribuição é ser um grupo consultivo para nossa CTO Rachel Laycock.

O TAB atua examinando tópicos que afetam soluções de tecnologia e tecnologistas da Thoughtworks. Esta edição do Thoughtworks Technology Radar é baseada em uma reunião do TAB realizada em Bengaluru, em março de 2026.

Rachel Laycock (CTO)

Abdul Jeelani

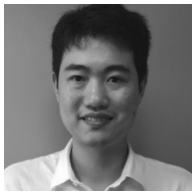
Alessio Ferri

Bharani Subramaniam

Birgitta Böckeler

Brandon Cook

Cecilia Geraldo



Chris Chakrit Riddhagni

Jagdsh LK Chand

Jim Gumbley

Effy Elden

Kief Morris

Ken Mugrage

May Xu

Nati Rivera

Ni Wang

Renan Martins

Nimisha Asthagiri

Selvakumar Natesan

Shangqi Liu

Sarang Sanjay Kulkarni

Vanya Seth

Créditos de produção

Equipe Editorial

- **Ken Mugrage**
  Technical Editor
- **Nati Rivera**
  Product Owner
- **Preeti Mishra**
  Project and Campaign Manager
- **Richard Gall**
  Content Editor
- **Michael Koch**
  Copy Editor
- **Gareth Morgan**
  Head of Content and Thought Leadership

Design e Multimídia

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

Experiência Digital

- **Rashmi Naganur**
  Business Analyst
- **Brigitte Britten-Kelly**
  Digital Content Strategist
- **Vandita Kamboj**
  UX Designer
- **Lohith Amruthappa**
  Analytics Specialist
- **Neeti Thakur**
  Marketing Automation Specialist

Comunicação

- **Shalini Jagadish**
  Internal Communications Specialist
- **Yasmin Brussulo**
  Internal Communications Specialist
- **Hiral Shah**
  Social Media Specialist
- **Abhishek Kasegaonkar**
  Social Media Specialist
- **Michelle Surendran**
  Public Relations Specialist
- **Anushree Tapuriah**
  Campaigns and Advertisement Specialist
- **Prakhar Nigam**
  Campaigns and Advertisement Specialist

Tradução — Português

- Ayrton Araujo
- Ignacio Morales
- Gustavo de Paula Andrade
- Renato Honjo
- Renata Silva
- Giovanna Scalfoni
- Ana Carolina Ramos
- Mayck Xavier
- Bruce Moares dos Santos
- Edney Filho
- Camila Guimarães
- Pedro Chiossi
- Carol Soares
- Thiago Gregorio

Créditos de produção

Contribuições de Thoughtworkers

Índia

- Krishnaswamy Subramaniam
- Rahul Garg
- OTTO account team
- Sayeed Hussain
- Srihari Sridharan
- Vijay Raghavan
- Jigar Jani

Europa

- David Rubio
- Jose Pech
- Florian Sellmayr
- Chris Ford
- Ricardo Piccoli
- Ben O'Mahony
- Dan Mutton
- Kai Hendry
- Karsten Lettow
- Alexandra Lovin
- Rieke Heinze
- Jose Maria Segui Gomez de Olea
- Tom Coggrave
- Muhammedsameer Azazi
- Kushwant Kumar
- Sivaprakash Kumar

APAC

- Hongbin Zhang
- Wei Feng
- Mi He
- Zhe Zhao
- Andrew Watson
- Zhe Zhao
- Daval Doshi
- Patiphon Puntusin
- Srini Raguraman
- Bosco Ho
- Dat Tran
- Arthur Nguyen

Americas

- Luiz Miccieli
- Carla Lima
- Gustavo Neves
- Ricardo Cavalcanti
- María Laura Jaramillo
- Gabriel Furini
- Guilherme Silveira
- Thiago Gomes
- Enderson Menezes
- Guilherme Marthe
- Daniel Romero
- Daniel Mansilla
- Chris Kramer
- Kurtis Angell
- Nikola Savic
- Jose Duron
- Cameron Casher
- Lauren O'Neal
- Brooks Bollich

Internal

- Johann Gomes
- Fabiano Damasceno
- Freddy Escobar
- Muhilvarnan V
- Rathinakumar Ponnusamy
- Da Cheng
- Sakthe Karthika T
- Vinodhini V

Temas

O desafio de avaliar tecnologia em um mundo de agentes

A IA está mudando não apenas a tecnologia, mas também como a analisamos e avaliamos. Ao montar esta edição do Technology Radar, notamos que avaliar tecnologia está se tornando mais difícil à medida que a indústria adota a IA. Um dos fatores que contribuem para isso é a difusão semântica: o rápido surgimento de novos termos para práticas em evolução, frequentemente antes que seus significados se estabilizem. Por exemplo, termos como o desenvolvimento orientado a especificações e a harness engineering às vezes são usados de forma inconsistente ou se sobrepõem em significado. Sem definições compartilhadas, é difícil determinar se estamos vendo técnicas distintas ou simplesmente rótulos diferentes para ideias semelhantes. Distinguir uma metodologia de engenharia madura e autônoma, com guardrails claras, do uso cotidiano de ferramentas de IA, como assistentes de programação, é uma dificuldade contínua.

O desafio vai além da semântica. O ritmo das mudanças agrava essa incerteza. Encontramos várias ferramentas com menos de um mês de existência — algumas promissoras, mas, em última análise, jovens demais para serem avaliadas. Em muitos casos, essas ferramentas eram mantidas por uma única pessoa contribuidora trabalhando com um agente de IA para programação. A IA diminuiu a barreira para a construção de ferramentas para pessoas desenvolvedoras, criando um fluxo constante de novas ferramentas. Isso força o ritmo tradicional do Radar: se dermos tempo para que as ferramentas amadureçam, nossas recomendações correm o risco de se tornar desatualizadas; se nos movermos rápido demais, corremos o risco de destacar tendências que desaparecem na mesma velocidade. Isso também levanta questões sobre sustentabilidade: quando algo pode ser criado rapidamente e com relativamente pouco esforço, o que garante o investimento contínuo para mantê-lo e evoluí-lo?

Esse ambiente também corre o risco de criar dívida cognitiva da base de código. À medida que mais código é gerado por IA, fica mais fácil adotar soluções sem desenvolver os modelos mentais necessários para entender como elas funcionam. Com o tempo, essa lacuna de entendimento se acumula, tornando os sistemas mais difíceis de entender, depurar e evoluir.

Mantendo princípios, abandonando padrões

Uma consequência interessante da IA no desenvolvimento de software é que ela não está apenas nos forçando a olhar para o futuro; ela também está nos levando a revisitar os fundamentos do nosso ofício. Ao montar esta edição, nos vimos retornando a muitas técnicas estabelecidas, da programação em par à arquitetura de confiança zero, e dos testes de mutação às métricas DORA. Revisitamos princípios centrais da maestria em software, como clean code, design intencional, testabilidade e acessibilidade como uma preocupação prioritária. Isso não é nostalgia, mas um contrapeso necessário à velocidade com que as ferramentas de IA podem gerar complexidade. Também observamos um

Temas

ressurgimento da linha de comando: após anos abstraindo-a em nome da usabilidade, as ferramentas baseadas em agentes estão trazendo as pessoas desenvolvedoras de volta ao terminal como uma interface principal.

Dito isso, esta não é apenas uma história de continuidade. O desenvolvimento de software assistido por IA representa uma mudança fundamental na prática de engenharia, exigindo que repensemos — e, em alguns casos, descartemos — suposições mantidas há muito tempo. Em particular, a forma como colaboramos e estruturamos os times precisará evoluir à medida que entendermos melhor o que é possível com sistemas baseados em agentes. Talvez precisemos considerar topologias de agentes em conjunto com as topologias de times, e repensar os ciclos de feedback de acordo. Várias técnicas nesta edição refletem essa mudança. Por exemplo, a medição da qualidade da colaboração com agentes de programação aponta para uma redefinição mais ampla do que significa ser uma pessoa desenvolvedora de software. Em última análise, provavelmente estamos apenas no começo de uma transformação mais profunda na forma como construímos, entendemos e interagimos com sistemas de software.

No centro disso está o desafio de gerenciar a dívida cognitiva em um ambiente impulsionado por IA. Devemos reter os princípios da boa engenharia de software sem permitir que as ferramentas de IA amplifiquem a complexidade ou acelerem a fadiga de decisão. Nossa tradição de craftsmanship há muito tempo defende que a velocidade sem disciplina multiplica os custos — um princípio ao qual vale a pena nos apegarmos à medida que os sistemas baseados em agentes tornam mais fácil do que nunca avançar rapidamente e entender o que construímos.

Protegendo agentes ávidos por permissões

"Ávido por permissões" descreve o dilema no centro do atual momento dos agentes: os agentes que valem a pena construir são aqueles que precisam de acesso a tudo. O OpenClaw e o Claude Cowork supervisionam tarefas de trabalho reais; o Gas Town coordena enxames de agentes em bases de código inteiras. Esses agentes exigem amplo acesso a dados privados, comunicação externa e sistemas reais — cada um argumentando que a recompensa justifica isso.

No entanto, como um esquiador que acabou de aprender a fazer curvas e se lança com confiança na pista mais difícil, as salvaguardas não acompanharam essa ambição. O apetite por acesso colide com problemas não resolvidos. A injeção de prompt significa que os modelos ainda não conseguem distinguir de maneira segura instruções confiáveis de entradas não confiáveis. A definição de "tríade letal" de Simon Willison para um agente inseguro — dados privados, conteúdo não confiável, ação externa — agora descreve a maioria dos agentes úteis por padrão, não por erro de configuração. A injeção não é a única ameaça. O comportamento do modelo ainda é inconsistente: uma tarefa concluída com sucesso uma vez não oferece garantia de que será bem-sucedida na próxima, muito

menos em escala. Os agentes encontram caminhos criativos de exfiltração, fazem push para branches que não deveriam tocar e enfraquecem os pontos de controle de aprovação/negação sem nenhuma intenção maliciosa ou prompting explícito.

O que podemos fazer hoje? Confiança zero, privilégio mínimo, melhorias de modelo e defesa em profundidade agora são requisitos básicos, mas não há bala de prata. Esperamos que sistemas de agentes seguros sejam compostos não por agentes monolíticos, mas por pipelines de agentes mais restritos, com forte monitoramento e controle. Práticas emergentes, a exemplo de skills de agentes como uma alternativa controlada ao MCP, agentes duráveis e técnicas para prevenir o inchaço de instruções de agentes, apontam todas nessa direção. O espaço está evoluindo rapidamente, o que é empolgante — mas, por enquanto, a cautela é essencial para evitar um erro que custe caro.

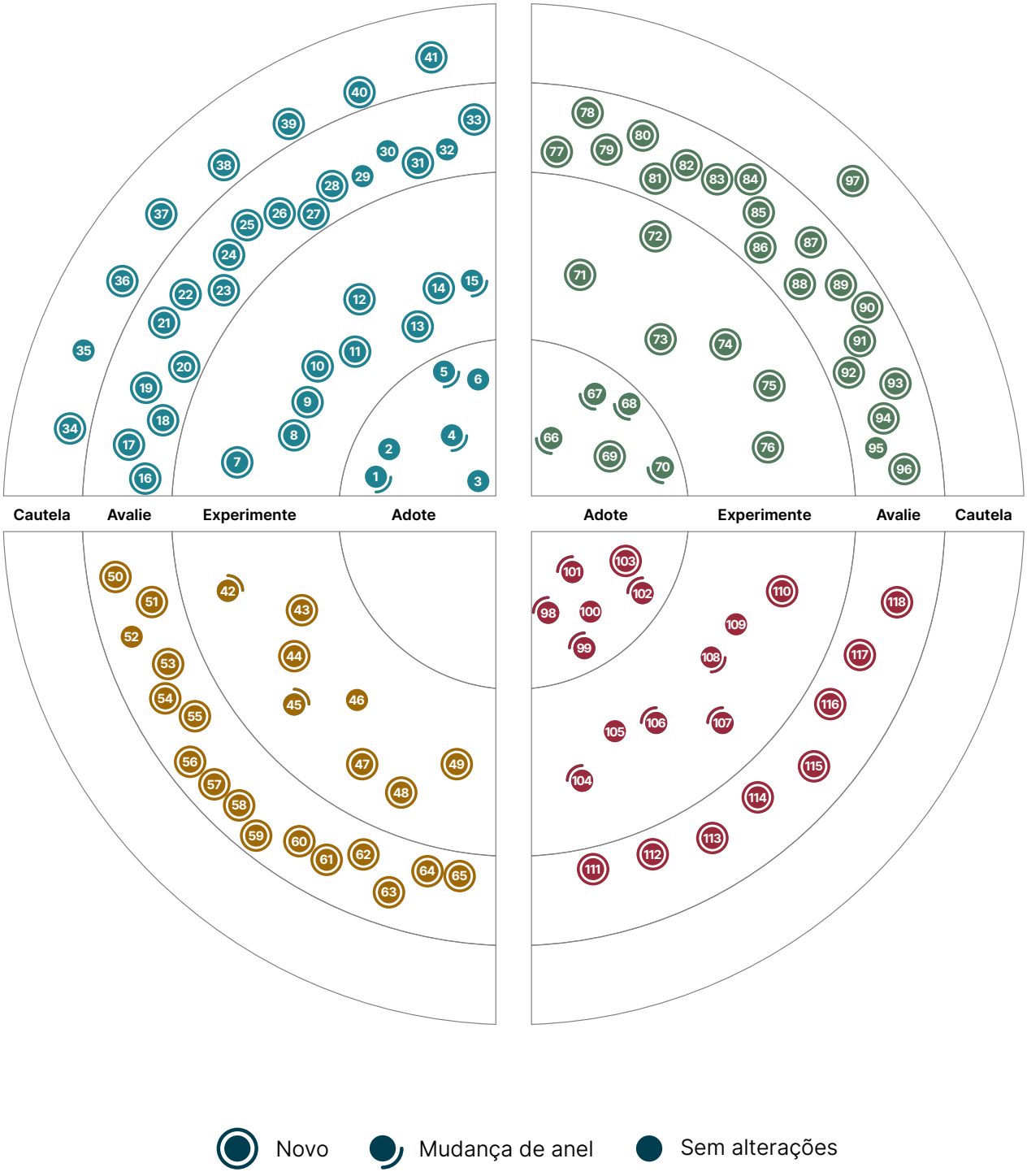
Mantendo agentes de programação sob controle

À medida que os agentes de programação se tornam mais capazes e produzem melhores resultados, os humanos estão cada vez mais tentados a sair do loop. Para mitigar os riscos da redução da supervisão humana, os times estão começando a investir em harnesses para agentes de programação: controles que guiam o comportamento dos agentes antes que o código seja gerado e fornecem feedback depois, para permitir a autocorreção.

Controles de antecipação (feedforward) antecipam o que o agente precisa para aumentar a probabilidade de acertar já na primeira tentativa. As skills de agentes têm sido um grande avanço recente, ajudando a modularizar instruções e convenções, e a carregá-las sob demanda. O Superpowers é um exemplo de um catálogo de skills úteis para times de software, e o conceito emergente de marketplaces de plugins está tornando mais fácil distribuir skills e outras configurações de contexto. Nossos times também viram valor em frameworks de desenvolvimento orientado a especificações, como o GitHub Spec-Kit e o OpenSpec, que estruturam workflows e guiam os agentes através do planejamento, design e implementação.

Controles de feedback, por outro lado, observam o comportamento após o agente agir e criam um ciclo para autocorreção. Essa abordagem é capturada pelos sensores de feedback para agentes de programação: quality gates determinísticos — compiladores, linters, verificadores de tipos e suítes de testes — integrados diretamente aos workflows dos agentes, para que falhas disparem a correção automática antes da revisão humana. Exemplos deste Radar incluem o cargo-mutants e outras ferramentas de testes de mutação, ferramentas de fuzz testing como o WuppieFuzz e ferramentas de análise de qualidade de código como o CodeScene. Além do feedback no loop, alguns de nossos times também reduziram o desvio de arquitetura combinando regras estruturais determinísticas com avaliação baseada em LLM.

O Radar



Técnicas

Adote

1. Engenharia de contexto
2. Instruções compartilhadas com curadoria para times de software
3. Métricas DORA
4. Passkeys
5. Saídas estruturadas de LLMs
6. Arquitetura de confiança zero

Experimente

7. Skills de agentes
8. Testes de componentes baseados em navegador
9. Sensores de feedback para agentes de programação
10. Mapeamento de code smells para técnicas de refatoração
11. Testes de mutação
12. Revelação progressiva de contexto
13. Execução em sandbox para agentes de programação
14. Camada semântica
15. UI orientada a servidor

Avalie

16. Ambientes de aprendizado por reforço baseados em agentes
17. Architecture drift reduction with LLMs
18. Inteligência de código como ferramenta para agentes
19. Context graph
20. Flywheel de feedback
21. HTML Tools
22. Avaliação de LLMs usando entropia semântica
23. Medindo a qualidade da colaboração com agentes de programação
24. MITRE ATLAS
25. Ralph Loop
26. Engenharia reversa de design systems
27. Isolamento contextual baseado em papéis no RAG
28. Skills como documentação executável de onboarding
29. Modelos de linguagem de pequeno porte
30. Time de agentes de programação
31. Fakes temporais
32. Análise de fluxo tóxico para IA
33. Modelos de linguagem visual para parsing de documentos de ponta a ponta

Cautela

34. Inchaço de instruções para agentes
35. Shadow IT acelerada por IA
36. Dívida cognitiva da base de código
37. Enxames de agentes de programação
38. Taxa de transferência de programação como medida de produtividade
39. Ignorar a durabilidade em workflows de agentes
40. MCP por padrão
41. Ambientes de desenvolvimento com streaming de pixels

Plataformas

Adote

—

Experimente

42. AG-UI Protocol
43. Apache APISIX
44. AWS Bedrock AgentCore
45. Graphiti
46. Langfuse
47. Port
48. Replit
49. SigNoz

Avalie

50. Agent Trace
51. ClickStack
52. Coder
53. Databricks Agent Bricks
54. DuckLake
55. FalkorDB
56. Google Dialogflow CX
57. MCP Apps
58. Monarch
59. Neutree
60. OptScale
61. Rhesis
62. RunPod
63. Sprites
64. torchforge
65. torchtitan

Cautela

—

Ferramentas

Adote

66. Axe-core
67. Claude Code
68. Cursor
69. Kafbat UI
70. mise

Experimente

71. cargo-mutants
72. Claude Code plugin marketplace
73. Dev Containers
74. Figma Make
75. OpenAI Codex
76. Typst

Avalie

77. Agent Scan
78. Beads
79. Bloom
80. CDK Terrain
81. CodeScene
82. ConfIT
83. Entire CLI
84. Git-AI
85. Google Antigravity
86. Google Mainframe Assessment Tool
87. OpenCode
88. OpenSpec
89. PageIndex
90. Pencil
91. Pi
92. Qwen 3 TTS
93. SGLang
94. ty
95. Warp
96. WuppieFuzz

Cautela

97. OpenClaw

Linguagens e Frameworks

Adote

98. Apache Iceberg
99. Declarative Automation Bundles
100. React JS
101. React Native
102. Svelte
103. Typer

Experimente

104. Agent Development Kit (ADK)
105. DeepEval
106. Docling
107. LangExtract
108. LangGraph
109. LiteLLM
110. Modern.js

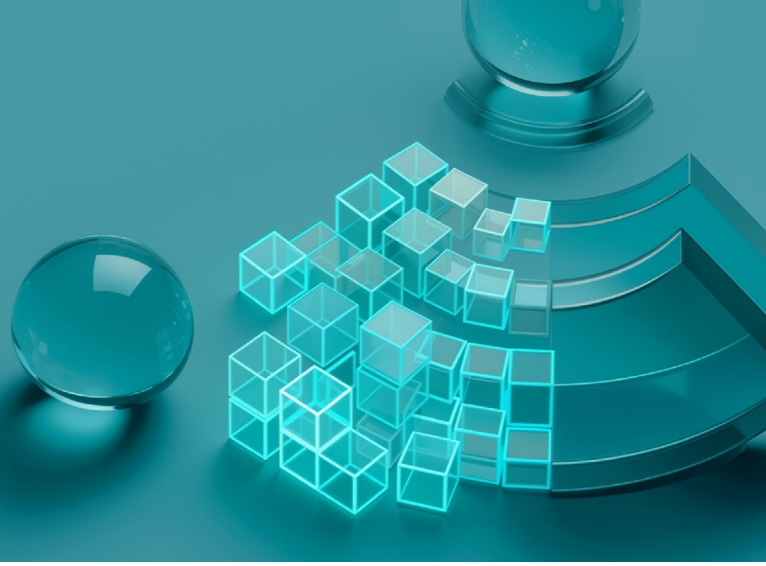
Avalie

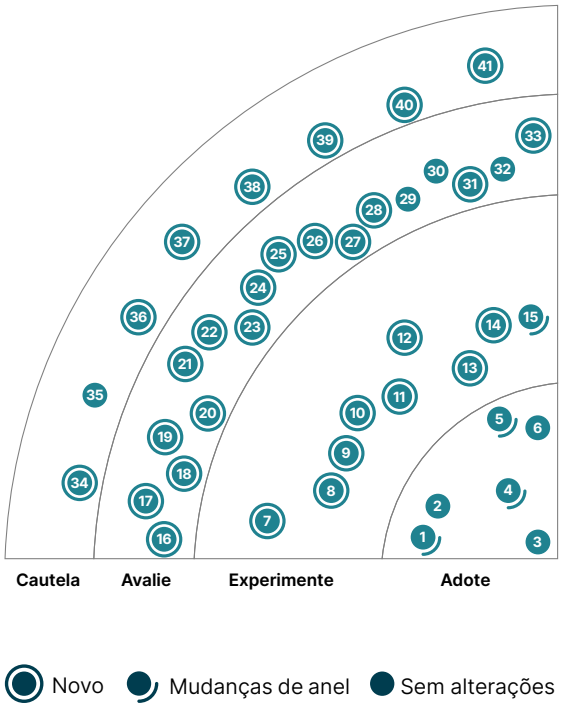
111. Agent Lightning
112. GitHub Spec Kit
113. Mastra
114. Pipecat
115. Superpowers
116. TanStack Start
117. TOON (Token-Oriented Object Notation)
118. Unsloth

Cautela

—

Técnicas





Novo Mudanças de anel Sem alterações

Adote

1. Engenharia de contexto
2. Instruções compartilhadas com curadoria para times de software
3. Métricas DORA
4. Passkeys
5. Saídas estruturadas de LLMs
6. Arquitetura de confiança zero

Experimente

7. Skills de agentes
8. Testes de componentes baseados em navegador
9. Sensores de feedback para agentes de programação
10. Mapeamento de code smells para técnicas de refatoração
11. Testes de mutação
12. Revelação progressiva de contexto
13. Execução em sandbox para agentes de programação
14. Camada semântica
15. UI orientada a servidor

Avalie

16. Ambientes de aprendizado por reforço baseados em agentes
17. Architecture drift reduction with LLMs
18. Inteligência de código como ferramenta para agentes
19. Context graph
20. Flywheel de feedback
21. HTML Tools
22. Avaliação de LLMs usando entropia semântica
23. Medindo a qualidade da colaboração com agentes de programação
24. MITRE ATLAS
25. Ralph Loop
26. Engenharia reversa de design systems
27. Isolamento contextual baseado em papéis no RAG
28. Skills como documentação executável de onboarding
29. Modelos de linguagem de pequeno porte
30. Time de agentes de programação
31. Fakes temporais
32. Análise de fluxo tóxico para IA
33. Modelos de linguagem visual para parsing de documentos de ponta a ponta

Cautela

34. Inchaço de instruções para agentes
35. Shadow IT acelerada por IA
36. Dívida cognitiva da base de código
37. Enxames de agentes de programação
38. Taxa de transferência de programação como medida de produtividade
39. Ignorar a durabilidade em workflows de agentes
40. MCP por padrão
41. Ambientes de desenvolvimento com streaming de pixels

1. Engenharia de contexto

Adote

A engenharia de contexto evoluiu de uma tática de otimização para uma preocupação arquitetural fundamental nos sistemas de IA modernos. Diferentemente da engenharia de prompt, que foca na formulação das palavras, a engenharia de contexto trata a janela de contexto como uma superfície de design e constrói intencionalmente o ambiente de informação da IA. À medida que os agentes lidam com tarefas mais complexas, despejar dados brutos em grandes janelas de contexto leva à "degradação de contexto" e a um raciocínio degradado. Para combater isso, os times estão mudando de prompts estáticos e monolíticos para a divulgação progressiva de contexto. Em vez de carregar antecipadamente todas as instruções e referências de que um agente possa precisar, esses sistemas começam com um índice leve do que está disponível. O agente determina quais prompts ou contextos são relevantes e puxa apenas o que é necessário, mantendo a relação sinal-ruído afiada a cada passo. Estamos vendo várias técnicas amadurecerem neste espaço: A configuração de contexto aproveita o cache de prompt para carregar instruções estáticas antecipadamente, reduzindo custos e melhorando o tempo até o primeiro token. A recuperação dinâmica vai além do RAG básico ao selecionar ferramentas e carregar apenas os servidores MCP necessários, evitando a expansão desnecessária do contexto. Os grafos de contexto modelam o raciocínio institucional — como políticas, exceções e precedentes — como dados estruturados e consultáveis. As técnicas de gerenciamento de contexto usam compressão stateful e subagentes para sumarizar saídas intermediárias em workflows de longa duração. Tratar o contexto da IA como uma caixa de texto estática é um caminho rápido para alucinações. Para construir agentes corporativos resilientes, os times devem projetar a engenharia de contexto como um pipeline dinâmico e gerenciado com precisão.

2. Instruções compartilhadas com curadoria para times de software

Adote

À medida que os times amadurecem no uso da IA, depender de pessoas desenvolvedoras individuais para escrever prompts do zero está surgindo como um antipadrão. Defendemos as instruções compartilhadas com curadoria para times de software, tratando as orientações da IA como um ativo colaborativo de engenharia em vez de um workflow pessoal. Inicialmente, essa prática focava em manter bibliotecas de prompts de uso geral para tarefas comuns. Agora estamos vendo uma evolução mais eficaz especificamente para ambientes de programação: ancorar essas instruções diretamente nos templates de serviços. Ao colocar arquivos de instrução como `CLAUDE.md`, `AGENTS.md` ou `.cursorrules` no repositório base usado para estruturar novos serviços, o template se torna um poderoso mecanismo de distribuição para a orientação da IA. Durante nossas discussões do Radar, também exploramos uma prática relacionada: ancorar agentes de programação a uma aplicação de referência. Aqui, uma base de código ativa e compilável serve como fonte de verdade. À medida que a arquitetura e os padrões de código evoluem, tanto a aplicação de referência quanto as instruções embutidas podem ser atualizadas. Novos repositórios herdam, então, os workflows e regras mais recentes dos agentes por padrão. Essa abordagem garante que uma assistência de IA consistente e de alta qualidade seja incorporada a cada projeto desde o primeiro dia, enquanto separa as bibliotecas de prompts gerais da configuração de IA específica do repositório.

3. Métricas DORA

Adote

As métricas definidas pelo programa de pesquisa DORA têm sido amplamente adotadas e provaram ser fortes indicadores de desempenho de uma organização de entrega. Essas métricas incluem lead time para mudanças, frequência de implantação, tempo médio de recuperação (MTTR), taxa de falha em mudanças e a mais nova quinta métrica, a taxa de retrabalho. A taxa de retrabalho é

uma métrica de estabilidade que mede quanto do pipeline de entrega de um time é consumido por retrabalho não planejado para corrigir um trabalho considerado concluído anteriormente, como bugs ou defeitos voltados para a pessoa usuária. Na era do desenvolvimento de software assistido por IA, as métricas DORA são mais importantes do que nunca. Medir a produtividade por linhas de código geradas por IA é enganoso; a melhoria real deve ser refletida no fluxo de entrega e na estabilidade. Se os lead times não diminuírem e a frequência de implantação não aumentar, a geração de código mais rápida não se traduzirá em melhores resultados. Por outro lado, a degradação nas métricas de estabilidade — particularmente a taxa de retrabalho — serve como um sinal de alerta precoce sobre pontos cegos, dívida técnica e os riscos do desenvolvimento assistido por IA sem supervisão. Como sempre, recomendamos usar essas métricas para reflexão e aprendizado do time, em vez de apenas construir dashboards complexos. Mecanismos simples, como check-ins durante retrospectivas, costumam ser mais eficazes para melhorar as capacidades do que ferramentas de rastreamento excessivamente detalhadas.

4. Passkeys

Adote

Guiadas pela FIDO Alliance e apoiadas por Apple, Google e Microsoft, as passkeys amadureceram para Adotar (Adopt). Elas são credenciais FIDO2 que podem substituir senhas usando criptografia de chave pública assimétrica. A chave privada é armazenada em um enclave seguro apoiado por hardware no dispositivo da pessoa usuária, protegida por biometria ou um PIN, e nunca sai de lá. Cada credencial é vinculada à origem do seu domínio na parte confiante (relying-party), tornando as chaves de acesso estruturalmente resistentes a phishing: um site sósia não recebe nada, diferente dos códigos OTP por SMS ou TOTP que um proxy de phishing pode interceptar. Com o phishing sendo responsável por mais de um terço de todas as violações de dados, essa resistência estrutural é cada vez mais importante. O FIDO Alliance Passkey Index 2025 relata que existem mais de 15 bilhões de contas elegíveis globalmente, o Google relata uma melhoria de 30% nas taxas de sucesso de login em 800 milhões de pessoas usuárias e a Amazon registrou logins seis vezes mais rápidos do que usando métodos tradicionais. A NIST SP 800-63-4 (julho de 2025) agora classifica as chaves de acesso sincronizadas como em conformidade com AAL2, revertendo orientações anteriores, e reguladores nos Emirados Árabes Unidos, Índia e agências federais dos EUA exigem autenticação resistente a phishing para serviços financeiros e sistemas governamentais. O FIDO Credential Exchange Protocol permite a portabilidade segura de chaves de acesso entre gerenciadores de credenciais, resolvendo preocupações anteriores de vendor lock-in. Os principais provedores de identidade, incluindo Auth0, Okta e Azure AD, agora suportam chaves de acesso como um recurso de primeira classe, e a implementação foi simplificada de um esforço de vários meses para um projeto de duas sprints. Nós adotamos chaves de acesso internamente e as tratamos como o ponto de partida padrão para novas implementações de autenticação. Os times devem projetar a recuperação de contas com cuidado e evitar caminhos de fallback sujeitos a phishing, como SMS OTP, que reintroduz as vulnerabilidades que as chaves de acesso eliminam. Credenciais vinculadas ao dispositivo em chaves de segurança de hardware continuam sendo necessárias para cenários AAL3, como acesso privilegiado.

5. Saídas estruturadas de LLMs

Adote

Saída estruturada de LLMs é a prática de restringir modelos para produzir respostas em um formato pré-definido, como JSON ou uma classe de linguagem de programação específica. Continuamos a ver essa técnica entregar resultados confiáveis em produção e agora nós a consideramos um padrão sensato para aplicações que consomem respostas de LLM de forma programática. Todos os principais provedores de modelos de IA agora oferecem modos nativos para saídas estruturadas, mas as implementações diferem nos subconjuntos de esquemas JSON que suportam, e essas APIs continuam

a evoluir rapidamente. Recomendamos o uso de bibliotecas como Instructor ou frameworks como Pydantic AI para fornecer abstrações estável entre provedores, com validação e novas tentativas automáticas, ou Outlines para geração restrita em modelos auto-hospedados.

6. Arquitetura de confiança zero

Adote

À medida que entramos na era dos agentes, muitas empresas estão lidando com o desafio de como construí-los e, ao mesmo tempo, endereçar os riscos de segurança que surgem ao conceder autonomia a sistemas imprevisíveis. A arquitetura de confiança zero (zero trust architecture - ZTA) continua sendo um padrão recomendado para construir e operar agentes com segurança. Princípios como "nunca confie, sempre verifique", junto com segurança baseada em identidade e acesso de privilégio mínimo, devem ser tratados como fundamentais para qualquer implantação de agentes. Nossos times estão aplicando padrões como o SPIFFE a agentes, estabelecendo bases sólidas de identidade e permitindo autenticação refinada em ambientes dinâmicos. O monitoramento contínuo e a verificação do comportamento dos agentes também são essenciais para o gerenciamento proativo de ameaças. Além das implantações de agentes, nossos times estão adotando práticas como OIDC impersonation no GCP para diferentes aplicações, incluindo pipelines de CI/CD, substituindo chaves estáticas de longa duração por tokens de curta duração emitidos após a verificação de identidade. Recomendamos que os times tratem os princípios da ZTA como padrões inegociáveis, independentemente do sistema que esteja sendo construído.

7. Skills de agentes

Experimente

À medida que os agentes de IA evoluem de simples interfaces de chat para a execução autônoma de tarefas, a engenharia de contexto se tornou um desafio crítico. As Skills de agentes fornecem um padrão aberto para modularizar o contexto ao empacotar instruções, scripts executáveis e recursos associados, como documentação. Os agentes carregam as Skills apenas quando necessário, com base em suas descrições, o que reduz o consumo de tokens e mitiga o esgotamento da janela de contexto e problemas como o inchaço de instruções de agentes. As Skills foram adotadas muito rapidamente, não apenas em agentes de programação, mas também em assistentes pessoais como o OpenClaw. Elas também são um dos motivos pelos quais os times estão se tornando mais cautelosos em relação a usar o MCP por padrão, já que muitos casos de uso podem ser resolvidos com a mesma eficácia apontando um agente para uma CLI ou script local. À medida que as Skills de agentes cresceram em popularidade, o ecossistema ao redor também se expandiu. Os marketplaces de plugins estão surgindo como uma maneira de versionar e compartilhar Skills, e múltiplos esforços estão explorando como avaliar a eficácia das skills. Nós, no entanto, alertamos contra o reuso não revisado e não verificado de Skills de terceiros, pois elas introduzem sérios riscos de segurança na cadeia de suprimentos.

8. Testes de componentes baseados em navegador

Experimente

No passado, ao discutir testes de componentes, nós geralmente desaconselhávamos ferramentas baseadas em navegador. Elas eram difíceis de configurar, lentas para rodar e frequentemente instáveis. Isso melhorou significativamente. Hoje, os testes de componentes baseados em navegador, usando ferramentas como o Playwright, são uma abordagem viável e muitas vezes preferível. Rodar testes em um navegador real fornece mais consistência, já que o teste equivale ao ambiente onde o código realmente é executado. O impacto na performance agora é pequeno o suficiente para que a contrapartida valha a pena. A instabilidade também diminuiu, e estamos vendo mais valor do que em ambientes emulados como o jsdom.

9. Sensores de feedback para agentes de programação

Experimente

Para tornar os agentes de programação mais eficazes e reduzir a carga sobre os revisores humanos, os times precisam de ciclos de feedback que os agentes possam acessar diretamente. Esses sensores de feedback para agentes de programação agem como uma forma de backpressure, aumentando a confiança nos resultados gerados. As pessoas desenvolvedoras há muito tempo dependem de quality gates determinísticos, como compiladores, linters, testes estruturais e suítes de testes; aqui, eles são integrados aos workflows baseados em agentes de modo que as falhas acionem uma autocorreção oportuna. Essas verificações reduzem o trabalho de direcionamento rotineiro para a pessoa no loop. Os times podem implementá-los de maneiras diferentes, como introduzindo um agente revisor responsável por executar verificações e acionar correções, ou expondo as verificações por meio de um processo complementar em execução paralela que os agentes podem consultar com eficiência. Agentes de programação também tornam mais barato construir linters personalizados e testes estruturais, fortalecendo ainda mais esses ciclos de feedback. Sempre que possível, esses sensores devem ser executados durante a sessão de programação e relatar resultados limpos antes que um commit seja criado, em vez de depender de verificações pós-commit.

10. Mapeamento de code smells para técnicas de refatoração

Experimente

Mapear code smells para técnicas de refatoração significa instruir um agente a lidar com problemas específicos com uma abordagem definida. A primeira camada tipicamente aponta o agente para uma referência genérica, como o livro *Refactoring*, para casos comuns. Para problemas mais especializados, os times podem mapear smells únicos para técnicas específicas usando Skills de agentes, comandos com barra (/) ou o AGENTS.md. Quando integrado com ferramentas de linting, isso cria um feedback determinístico, acionando a abordagem de refatoração apropriada sempre que um smell é detectado. Isso é particularmente eficaz para stacks legadas como .NET Framework 2.0 ou Java 8, onde os dados de treinamento genéricos frequentemente deixam a desejar. Também é útil para times com padrões de engenharia distintos. Sem essas instruções direcionadas, um agente tenderá a adotar padrões genéricos por padrão, em vez de seguir requisitos específicos.

11. Testes de mutação

Experimente

Os testes de mutação continuam sendo o sinal mais honesto para avaliar a real capacidade de detecção de falhas de um conjunto de testes. Diferente da cobertura de código tradicional, que rastreia apenas a execução das linhas do código, essa técnica introduz bugs deliberados, ou mutações, no código-fonte para verificar se os testes falham quando o comportamento quebra. Se uma mutação passa despercebida, isso revela uma lacuna na validação em vez de apenas uma falta de cobertura. Essa distinção é crítica em uma era de desenvolvimento assistido por IA, onde altas porcentagens de cobertura podem mascarar testes logicamente vazios ou códigos gerados que nunca foram validados de forma significativa. Com os casos de teste gerados por IA sendo comuns atualmente, os testes de mutação agem como uma camada de reforço para capturar testes "perpetuamente verdes" — aqueles que passam nos testes independentemente de mudanças lógicas devido à falta de verificações (asserts) ou uso de mocks desacoplados. Ao usar ferramentas como o Stryker, Pitest ou cargo-mutants, mudamos o foco de quanto código é executado para quanto código é realmente verificado, particularmente na lógica central do domínio. O objetivo é garantir que um conjunto de testes passando (passing test suite) seja um sinal confiável da parte funcional estar correta, em vez de simplesmente um relatório de quais linhas foram executadas.

12. Revelação progressiva de contexto

Experimente

A revelação progressiva de contexto é uma técnica dentro da prática de engenharia de contexto. Em vez de sobrecarregar um agente com instruções logo de cara, você fornece a ele uma fase leve de descoberta na qual ele seleciona o que precisa com base no prompt da pessoa usuária, carregando informações detalhadas na janela de contexto apenas quando elas se tornam relevantes. Isso funciona muito bem para cenários de RAG, onde um agente primeiro identifica o domínio relevante a partir das consultas das pessoas usuárias e, em seguida, recupera instruções e dados específicos de acordo. É também assim que muitas ferramentas de programação baseadas em agentes lidam com as Skills de agentes, determinando primeiro quais skills são relevantes para uma tarefa antes de carregar instruções detalhadas, em vez de fornecer um conjunto de instruções monolítico e único, cheio de condições e ressalvas. Ao construir sistemas baseados em agentes, é fácil cair na armadilha do inchaço de instruções com infinitas regras do que fazer e não fazer na tentativa de controlar o comportamento, o que pode, em última análise, degradar o desempenho. A divulgação progressiva de contexto evita isso ao garantir que o agente receba a orientação certa no momento certo, mantendo a janela de contexto enxuta e prevenindo a degradação de contexto.

13. Execução em sandbox para agentes de programação

Experimente

Execução em sandbox para agentes de programação é a prática de executar agentes dentro de ambientes isolados com acesso restrito ao sistema de arquivos, conectividade de rede controlada e uso de recursos limitado. À medida que agentes de programação ganham autonomia para executar código, rodar builds e interagir com o sistema de arquivos, conceder acesso irrestrito a um ambiente de desenvolvimento introduz riscos reais, desde danos acidentais até exposição de credenciais. Enxergamos o sandboxing como um padrão sensato por padrão, e não como um aprimoramento opcional.

O leque de opções de sandboxing agora abrange um amplo espectro. Num extremo, muitos agentes de programação oferecem modos sandbox integrados, e Dev Containers fornecem isolamento familiar baseado em contêiner. No outro, ferramentas dedicadas assumem posições diferentes no trade-off entre efêmero e persistente. Shuru sobe microVMs descartáveis que reiniciam a cada execução, enquanto Sprites oferece ambientes com estado com checkpoint e restore. Para isolamento nativo em Linux, Bubblewrap oferece sandboxing leve baseado em namespaces e, no macOS, `sandbox-exec` oferece proteção semelhante.

Além do isolamento básico, os times devem considerar os requisitos práticos de um sandbox produtivo. Isso inclui tudo o que é necessário para build e testes, bem como autenticação segura e simples com serviços como GitHub e provedores de modelo. Pessoas desenvolvedoras precisam de encaminhamento de portas e CPU e memória suficientes para cargas de trabalho de agentes. Se o sandbox deve ser efêmero por padrão ou persistente para recuperação de sessão é uma decisão de projeto que dependerá das prioridades do time em segurança, custo e continuidade do fluxo de trabalho.

14. Camada semântica

Experimente

A camada semântica é uma técnica de arquitetura de dados que introduz uma camada de lógica de negócios compartilhada entre os armazenamentos de dados e as aplicações consumidoras, incluindo ferramentas de inteligência de negócios (BI), agentes de IA e APIs. Ela centraliza definições de métricas, joins, regras de acesso e terminologia de negócios para que os consumidores tenham

definições compartilhadas. O conceito é anterior à stack de dados moderna, mas tem visto um interesse renovado com abordagens focadas em código, como as lojas de métricas. Sem uma camada semântica comum, a lógica de negócios se dispersa em tabelas ad-hoc de data warehouse, dashboards e aplicações downstream, enquanto as definições de métricas silenciosamente se divergem - o que é particularmente problemático quando usadas para apoiar decisões de negócio. Nossos times viram isso se tornar mais agudo com a IA baseada em agentes: usando LLMs para realizar traduções ingênuas de texto para SQL irá frequentemente produzir resultados incorretos, especialmente quando regras de negócio, como o reconhecimento de receita, existem fora do esquema. As plataformas de nuvem agora estão incorporando camadas semânticas diretamente: o Snowflake chama isso de Semantic Views e o Databricks chama de Metric Views. Ferramentas independentes, como o dbt MetricFlow e o Cube, fornecem uma camada portátil entre sistemas. O recente lançamento do Open Semantic Interchange (OSI) v1.0, apoiado por vários fornecedores, sinaliza uma crescente padronização e interoperabilidade entre plataformas de análise de dados, IA e inteligência de negócios (BI). O custo principal é o investimento inicial em modelagem de dados. Os times devem começar com um único domínio em vez de tentar um rollout em toda a empresa, pois implantações amplas frequentemente deixam relatórios legados rodando em paralelo com a nova camada, reintroduzindo definições inconsistentes.

15. UI orientada a servidor

Experimente

A UI orientada a servidor está retornando ao anel Trial, pois vemos mais times encurtando com sucesso o caminho para produção. Ao separar a renderização em um contêiner genérico enquanto fornece estrutura e dados por meio do servidor, os times de mobile podem contornar os longos ciclos de revisão das lojas de aplicativos para cada iteração. Vimos que isso melhora significativamente o time to market, com formatos baseados em JSON permitindo atualizações em tempo real. Embora tenhamos alertado anteriormente contra as “bagunças horrendas e excessivamente configuráveis” que frameworks proprietários podem criar, o surgimento de padrões mais estáveis de empresas como Airbnb e Lyft tem ajudado a reduzir a complexidade. Nossa experiência mostra que o investimento substancial exigido para um framework proprietário agora é mais fácil de justificar para aplicações de grande escala. No entanto, isso ainda exige um forte caso de negócios e engenharia disciplinada para evitar a criação de um “protocolo deus” que se torna difícil de manter. Para times que buscam reduzir a complexidade no lado do cliente, essa abordagem fornece uma maneira poderosa de escalar entre os times e sincronizar a lógica em múltiplas plataformas. Recomendamos aplicá-la a áreas altamente dinâmicas de uma aplicação, em vez de usá-la como um substituto geral para todo o desenvolvimento de UI.

16. Ambientes de aprendizado por reforço baseados em agentes

Avalie

Os ambientes de aprendizado por reforço baseados em agentes fornecem um campo de treinamento para agentes baseados em LLM, combinando o contexto, as ferramentas e o feedback para concluir tarefas de múltiplas etapas. Essa abordagem reformula o pós-treinamento de LLMs, passando de saídas simples de turno único (single-turn) para comportamentos baseados em agentes, como raciocínio e uso de ferramentas, com recompensas ou penalidades atribuídas a cada ação. Técnicas como o RLVR ajudam a garantir que essas recompensas sejam verificáveis e resistentes a manipulações. Laboratórios de pesquisa em IA estão atualmente impulsionando o desenvolvimento desses ambientes, particularmente para agentes de programação e de uso de computador. Um exemplo fora dos laboratórios de fronteira é o Cursor’s Composer, um modelo de programação especializado treinado dentro do ambiente de seu produto. As organizações que constroem sistemas

baseados em agentes devem considerar se a criação de ambientes de aprendizado por reforço poderia ajudar a treinar modelos mais capazes e específicos para o seu domínio. Configurar a infraestrutura necessária pode ser complexo. No entanto, frameworks e plataformas estão surgindo para simplificar o processo, incluindo o environments hub da Prime Intellect, o Agent Lightning e o NVIDIA NeMo Gym. Recomendamos explorar essa abordagem onde ela possa entregar modelos mais capazes e com melhor custo-benefício para o seu domínio.

17. Architecture drift reduction with LLMs

Avalie

O uso crescente de agentes de programação de IA pode acelerar a deriva da base de código e dos designs de arquitetura planejadas. Se não for controlado, esse desvio se acumula à medida que agentes e humanos replicam padrões existentes, inclusive os já degradados, criando um ciclo de feedback onde código ruim gera código ainda pior. Alguns de nossos times agora estão abordando a redução de desvio de arquitetura com LLMs. Essa abordagem combina ferramentas de análise determinística (como Spectral, ArchUnit ou Spring Modulith com avaliação baseada em LLMs para detectar violações estruturais e semânticas. Os LLMs são então usados para ajudar a corrigir esses problemas. Nossos times aplicaram essa abordagem para impor diretrizes de qualidade de API nos serviços e para definir zonas arquiteturais que orientam as melhorias geradas por agentes. Algumas lições aprendidas: assim como no linting tradicional, as varreduras iniciais podem trazer à tona um grande número de violações que exigem triagem e priorização, e os LLMs podem ajudar nesse processo. Manter pequenas e pontuais correções geradas por agentes facilita a revisão, e um ciclo de verificação adicional é essencial para confirmar que as mudanças melhoram o sistema em vez de introduzir regressões. Essa técnica estende a ideia de mecanismos de feedback para agentes de programação para os estágios finais do ciclo de vida de entrega. Como um time da OpenAI descreve, a redução de desvio funciona como uma espécie de “garbage collection”, refletindo a realidade de que entropia e deterioração surgem até mesmo em sistemas com fortes ciclos de feedback iniciais.

18. Inteligência de código como ferramenta para agentes

Avalie

Os LLMs processam código como um fluxo de tokens; eles não têm compreensão nativa de diagramas de chamada (call graphs), hierarquias de tipo ou relacionamentos de símbolos. Para a navegação de código, a maioria dos agentes de programação de hoje adotam por padrão a busca baseada em texto, o denominador comum mais poderoso entre todas as linguagens. Para refatorações que são um atalho rápido em uma IDE, os agentes precisam gerar múltiplos diffs textuais. Como resultado, os agentes acabam gastando tokens significativos reconstruindo informações que já existem na árvore sintática abstrata (abstract syntax tree - AST). A inteligência de código como ferramenta para agentes dão aos agentes acesso a ferramentas que estão cientes da AST, por exemplo, via Language Server Protocol (LSP). Por meio dessas integrações, os agentes podem executar operações como “encontrar todas as referências a este símbolo” ou “renomear este tipo em todos os lugares” como ações de primeira classe, em vez de depender de substituições de texto frágeis. Outra integração poderosa de inteligência de código são as ferramentas de “codemod” como o OpenRewrite, que opera na representação ainda mais rica de Lossless Semantic Tree (LST) do código. O resultado são menos edições com alucinações e menor consumo de tokens ao delegar tarefas apropriadas a ferramentas determinísticas. Claude Code, OpenCode e outros se integram a servidores LSP rodando localmente; a JetBrains fornece um servidor MCP que expõe capacidades de navegação e refatoração da IDE para agentes externos, enquanto o servidor MCP Serena oferece recuperação semântica de código e edição.

19. Context graph

Avalie

O Context Graph é uma técnica de representação de conhecimento onde decisões, políticas, exceções, precedentes, evidências e resultados são modelados como nós conectados de primeira classe em um grafo, estruturados para consumo por IA. Enquanto os sistemas de registro capturam o que aconteceu, um grafo de contexto captura o porquê, transformando o raciocínio institucional enterrado em threads do Slack, cadeias de aprovação e na cabeça das pessoas em uma estrutura consultável e legível por máquina. Isso é vital para a eficácia dos agentes; um agente lidando com uma exceção de desconto, por exemplo, não consegue determinar se isso reflete uma política vigente ou uma exceção pontual, e pode raciocinar incorretamente. Um grafo de contexto pode expor diretamente essa procedência, permitindo que os agentes percorram rastros de decisão, apliquem precedentes relevantes e raciocinem através de cadeias causais multi-hop. Diferentemente do GraphRAG, que é construído a partir de corpora de documentos estáticos, um grafo de contexto mantém a validade temporal em cada aresta, de modo que fatos superados são invalidados em vez de sobrescritos. Vale a pena avaliar os grafos de contextos para aplicações baseadas em agentes que exigem memória persistente entre sessões ou raciocínio de decisão rastreável.

20. Flywheel de feedback

Avalie

Equipes que trabalham com agentes de programação estão adotando cada vez mais fluxos de trabalho de desenvolvimento orientado a especificações. Quer utilizem um framework leve ou mais diretivo, esses fluxos geralmente seguem uma sequência semelhante de especificação → planejamento → implementação. O flywheel de feedback estende esse fluxo com uma etapa adicional focada em melhorar continuamente a estrutura (*harness*) do agente de programação. A abordagem é semelhante às retrospectivas: as equipes capturam sucessos e falhas durante uma sessão do agente de programação e os usam para melhorar a previsibilidade de sessões futuras, o que gera ganhos cumulativos ao longo do tempo. É uma metatécnica na qual um humano na supervisão (*human-on-the-loop*) se concentra em melhorar os controles proativos (*feedforward*), como instruções compartilhadas com curadoria, bem como sensores de feedback para agentes de programação. Nossas equipes consideram essa prática eficaz, pois é análoga à refatoração de código. O próximo nível se parece mais com um flywheel de feedback baseado em agentes, onde, com base no feedback acumulado, o próprio agente decide quais melhorias são necessárias. Por enquanto, no entanto, as equipes ainda precisam de um humano no circuito (*human-in-the-loop*) para evitar a degradação de contexto e um feedback ruidoso que poderia desviar o agente do seu objetivo. Sugerimos utilizar essa abordagem para avaliar toda a estrutura (*harness*) do agente de programação à medida que seu ambiente evolui, especialmente ao adotar novos modelos; o que funcionou com um modelo pode não ser necessário com o próximo.

21. HTML Tools

Avalie

Como as ferramentas agênticas facilitam a construção de pequenos utilitários específicos para tarefas, o principal desafio costuma ser como implantá-los e compartilhá-los. HTML Tools é uma abordagem em que um script ou utilitário compartilhável é empacotado como um único arquivo HTML. Você pode rodá-los diretamente em um navegador, hospedá-los em qualquer lugar ou simplesmente compartilhar o arquivo. Essa abordagem evita o overhead de distribuir ferramentas de CLI, o que exige o compartilhamento de binários ou o uso de gerenciadores de pacotes. Também é mais simples do que construir uma aplicação web completa com hospedagem dedicada. De uma perspectiva

de segurança, rodar arquivos não confiáveis ainda traz riscos, mas o sandbox do navegador e a capacidade de inspecionar o código-fonte fornecem alguma mitigação. Para utilitários leves, um único arquivo HTML oferece uma maneira altamente acessível e portátil de compartilhar ferramentas.

22. Avaliação de LLMs usando entropia semântica

Avalie

A confabulação, uma forma de alucinação em aplicações de perguntas e respostas (QA) com LLMs, é difícil de abordar com métodos de avaliação tradicionais. Uma abordagem usa a entropia da informação como uma medida de incerteza, analisando a variação lexical nas respostas geradas para a mesma entrada. A avaliação de LLMs usando entropia semântica estende essa ideia focando nas diferenças de significado em vez da variação superficial. Essa abordagem avalia o significado em vez de sequências de palavras, tornando-a aplicável a diversos conjuntos de dados e tarefas sem exigir conhecimento prévio. Ela se generaliza bem para tarefas não vistas, ajudando a identificar prompts com maior probabilidade de gerar confabulações e indicando quando é preciso ter cautela. Os resultados mostram que a entropia simples frequentemente falha em detectar confabulações, enquanto a entropia semântica é mais eficaz para filtrar afirmações falsas.

23. Medindo a qualidade da colaboração com agentes de programação

Avalie

Estamos vendo ganhos reais de produtividade ao usar agentes de programação, mas a maioria das métricas de avaliação ainda foca excessivamente na taxa de transferência de programação, como o tempo até o primeiro output, linhas de código geradas e tarefas concluídas. Medir a qualidade da colaboração com agentes de programação ajuda os times a evitar cair na "armadilha da velocidade", mudando o foco para quão bem humanos e agentes trabalham juntos. Métricas como a taxa de aceitação na primeira tentativa, ciclos de iteração por tarefa, retrabalho pós-merge, builds que falharam e a carga de revisão fornecem sinais mais significativos do que apenas a velocidade. Times que usam o Claude Code podem usar o comando `/insights` para gerar relatórios que mostram os acertos e gargalos das sessões com agentes. Nossos times também têm feito testes para acompanhar a taxa de aprovação na primeira tentativa ao usar um comando `/review` customizado. Na prática, ciclos de feedback mais curtos e menos builds quebrados indicam uma interação mais eficaz com os agentes de programação. Quando os times se encontram em repetidas idas e vindas com seus agentes, essas métricas destacam oportunidades para melhorar o flywheel de feedback. Recomendamos rastrear a qualidade da colaboração no nível do time, em vez do nível individual, juntamente com as Métricas DORA para ter uma visão mais completa da adoção de agentes de programação.

24. MITRE ATLAS

Avalie

Sistemas baseados em agentes e ferramentas de programação introduzem novas arquiteturas e ameaças de segurança emergentes. O MITRE ATLAS é uma base de conhecimento de táticas e técnicas adversariais direcionadas a sistemas de IA e ML. Mais focado do que o amplo framework MITRE ATT&CK e projetado para complementá-lo, o ATLAS fornece uma taxonomia de ameaças para pipelines de ML, aplicações de LLM e sistemas baseados em agentes. Descobrimos que, sem um vocabulário compartilhado, os riscos de segurança são frequentemente negligenciados ou reduzidos a um checklist. É aqui que o ATLAS pode ajudar. Como o framework é fundamentado em pesquisas sobre incidentes reais e padrões de tecnologia, a equipe pode usá-lo para apoiar a modelagem de ameaças. Os times também podem considerá-lo um complemento natural para frameworks de controle como o SAIF, ajudando a descrever o cenário de ameaças em constante evolução para sistemas de IA.

25. Ralph Loop

Avalie

O Ralph Loop (também chamado às vezes de Wiggum loop) é uma técnica de agente de programação autônomo onde um prompt fixo é fornecido a um agente em um loop infinito. Cada iteração começa com uma nova janela de contexto: o agente seleciona uma tarefa de uma especificação ou plano, implementa-a e o loop recomeça. O insight central é a simplicidade. Em vez de orquestrar times de agentes de programação ou enxames de agentes de programação, um único agente trabalha autonomamente em relação a uma especificação, com a expectativa de que a base de código convergirá para a especificação ao longo de iterações repetidas. Usar uma janela de contexto nova a cada iteração evita a degradação de qualidade que vem do contexto acumulado, embora com um custo significativo de tokens. Ferramentas como o goose implementaram o padrão, em alguns casos estendendo-o com revisão entre modelos entre as iterações.

26. Engenharia reversa de design systems

Avalie

As organizações frequentemente lutam com interfaces legadas fragmentadas onde o "padrão de design" existe apenas como uma coleção frouxa de páginas web desconexas, materiais de marketing e screenshots. Historicamente, auditar esses artefatos para estabelecer uma base unificada tem sido um processo manual e demorado. Com LLMs multimodais, essa extração agora pode ser automatizada, efetivamente fazendo a engenharia reversa de design systems a partir de assets visuais existentes. Ao alimentar sites, screenshots e fragmentos de UI em ferramentas especializadas ou modelos de IA com capacidade de visão, os times podem extrair tokens de design essenciais — como paletas de cores, escalas de tipografia e regras de espaçamento — e identificar padrões de componentes recorrentes. A IA então sintetiza esses dados visuais não estruturados em uma representação semântica e estruturada de um design system. Quando integrada a ferramentas como o Figma, essa saída pode acelerar significativamente a criação de uma biblioteca de componentes formalizada e de fácil manutenção. Além de reduzir o esforço em auditorias visuais, essa técnica pode servir como um trampolim para a construção de design systems prontos para IA. Para empresas sobrecarregadas pela dívida de design brownfield, o uso de IA para estabelecer um design system base é um ponto de partida prático antes de um redesign completo ou padronização do frontend.

27. Isolamento contextual baseado em papéis no RAG

Avalie

O isolamento contextual baseado em papéis no RAG é uma técnica arquitetural que move o controle de acesso da camada de aplicação para a camada de recuperação. Cada fragmento de dados, é tagueado com permissões baseadas em papéis no momento da indexação. No momento da consulta, o motor de recuperação restringe o espaço de busca com base na identidade autenticada da pessoa usuária, que é correspondida com os metadados em cada fragmento. Isso garante que o modelo de IA não possa acessar contexto não autorizado porque ele é filtrado na etapa de recuperação. Isso fornece uma base de "confiança zero" para bases de conhecimento internas. Como muitos bancos de dados vetoriais agora suportam filtragem de metadados de alto desempenho, como o Milvus ou serviços construídos no Amazon S3, tornou-se mais prático adotar essa técnica, mesmo para grandes bases de conhecimento.

28. Skills como documentação executável de onboarding

Avalie

Skills de agentes, instruções compartilhadas com curadoria e muitas outras técnicas de engenharia de contexto aparecem ao longo desta edição do Radar. Um caso de uso que queremos destacar no contexto da programação é o uso de skills como documentação executável de onboarding. Essa técnica se aplica a vários níveis. Dentro de uma base de código, uma skill `/_setup` pode assumir o papel conjunto de um script go.sh e de um arquivo README, combinando scripts com semântica executada por LLM para etapas que não podem ser roteirizadas. Ela também pode ir além do que um script pode fazer, levando em consideração dinamicamente o estado atual da base de código e do ambiente. Em segundo lugar, pessoas criadoras de bibliotecas e APIs podem fornecer skills para seus consumidores como parte de sua documentação por meio de registros de skills internos ou externos, como o Tessl. E, em terceiro lugar, descobrimos que isso é útil para o onboarding de times em plataformas internas, a fim de diminuir a barreira para o uso de uma tecnologia-chave ou reduzir o atrito na adoção de um design system. Até agora, nossa experiência com isso dependeu fortemente de servidores MCP, mas agora está mudando para skills. Assim como em outras formas de documentação, o desafio de manter isso atualizado não desaparece. No entanto, diferente da documentação estática, a documentação executável pode ajudar você a notar a desatualização muito mais cedo.

29. Modelos de linguagem de pequeno porte

Avalie

Os modelos de linguagem de pequeno porte (SLMs) continuam a melhorar e estão começando a oferecer melhor inteligência por dólar do que os LLMs para certos casos de uso. Temos visto times avaliarem os SLMs para reduzir custos de inferência e acelerar workflows baseados em agentes. Progressos recentes mostram ganhos constantes na densidade de inteligência, tornando os SLMs competitivos com LLMs mais antigos para tarefas como sumarização e programação básica. Essa mudança reflete um afastamento do “maior é melhor” em direção a dados de maior qualidade, destilação de modelo e quantização. Modelos como Phi-4-mini e Ministral 3 3B demonstram como modelos destilados podem reter muitas capacidades de modelos professores maiores. Mesmo modelos ultracompactos, como Qwen3-0.6B e Gemma-3-270M, estão se tornando viáveis para a execução de modelos em dispositivos de borda. Para casos de uso baseados em agentes onde LLMs mais antigos têm sido suficientes, os times devem considerar os SLMs como uma alternativa de menor custo e menor latência com requisitos de recursos reduzidos.

30. Time de agentes de programação

Avalie

No Radar anterior, descrevemos um time de agentes de programação como uma técnica onde uma pessoa desenvolvedora orquestra um pequeno conjunto de agentes com funções específicas para colaborar em uma tarefa de programação. Desde então, a barreira de adoção diminuiu. O suporte a subagentes tornou-se um requisito básico entre as ferramentas de agentes de programação consolidadas, e o Claude Code agora inclui um recurso de equipes de agentes que oferece orquestração integrada. Em um time de agentes, um orquestrador principal tipicamente coordena o sequenciamento de tarefas e a paralelização. Os agentes devem ser capazes de se comunicar não apenas com o orquestrador, mas também entre si. Casos de uso comuns incluem times de agentes revisores ou grupos de agentes implementadores responsáveis por diferentes partes da aplicação,

como backend e frontend. Embora algumas pessoas na indústria estejam usando os termos “times de agentes” e “enxames de agentes” de forma intercambiável (por exemplo, o Claude Code descreve seu recurso de times de agentes como “nossa implementação de enxames”), vemos valor em distingui-los. Um time pequeno e intencional de agentes colaborando em uma tarefa difere significativamente de um grande enxame em termos de barreiras de entrada, complexidade e casos de uso.

31. Fakes temporais

Avalie

Os fakes temporais estendem a ideia de simular sistemas do mundo real para desenvolvimento e testes, uma prática usada há muito tempo em plataformas industriais e de IoT. Com os agentes de programação de IA reduzindo o esforço necessário para construir tais simuladores, os times agora podem criar réplicas de alta fidelidade de dependências externas com muito mais facilidade. Diferentemente dos mocks tradicionais que retornam pares estáticos de solicitação-resposta, os fakes temporais mantêm máquinas de estados internas e modelam a evolução temporal de sistemas reais. Um de nossos times usou essa técnica ao desenvolver uma stack de observabilidade para grandes data centers de GPU, evitando a necessidade de adquirir hardware físico. Testar regras de alerta, dashboards e detecção de anomalias contra sistemas reais pode ser impraticável — por exemplo, superaquecer intencionalmente uma GPU para validar um alerta de estrangulamento térmico (thermal throttle). Em vez disso, o time construiu fakes para domínios de hardware como NVIDIA DCGM e fabric InfiniBand usando Go. Esses simuladores permitiram cenários de falha como estrangulamento térmico, tempestades de erros XID, instabilidade de link e falhas de fonte de alimentação (PSU) com intensidade e duração configuráveis, orquestrados por meio de uma stack process-compose. Um registro central definia os cenários de falha válidos, enquanto um servidor MCP expunha a injeção de cenários ao agente. O agente podia acionar falhas — por exemplo, injetando um estrangulamento térmico em uma GPU específica — e verificar se as métricas mudaram, os alertas dispararam e os dashboards foram atualizados conforme o esperado. Essa fidelidade temporal torna a técnica valiosa para testar sistemas complexos onde as falhas ocorrem em cascata. No entanto, os times devem garantir que os fakes permaneçam fiéis ao comportamento do mundo real; caso contrário, correm o risco de criar uma falsa confiança em pipelines automatizados.

32. Análise de fluxo tóxico para IA

Avalie

As capacidades dos agentes estão superando as práticas de segurança. Com o surgimento de agentes ávidos por permissões como o OpenClaw, os times estão cada vez mais implantando agentes em ambientes que os expõem à tríade letal: acesso a dados privados, exposição a conteúdo não confiável e a capacidade de se comunicar externamente. À medida que as capacidades crescem, a superfície de ataque se expande, expondo os sistemas a riscos como injeção de prompt e envenenamento de ferramentas. Continuamos a ver a análise de fluxo tóxico como a principal técnica para examinar sistemas agênticos e identificar caminhos de dados inseguros e potenciais vetores de ataque. Esses riscos não se limitam mais às integrações MCP; nossos times observaram padrões semelhantes em Agent Skills, onde um ator malicioso pode empacotar uma skill aparentemente útil que incorpora instruções ocultas para exfiltrar dados sensíveis. Recomendamos fortemente que os times que trabalham com agentes realizem a análise de fluxo tóxico e que usem ferramentas como o Agent Scan para identificar caminhos de dados inseguros antes que eles sejam explorados.

33. Modelos de linguagem visual para parsing de documentos de ponta a ponta

Avalie

O parsing de documentos frequentemente depende de pipelines de múltiplos estágios que combinam detecção de layout, OCR tradicional e scripts de pós-processamento. Essas abordagens costumam ter dificuldades com layouts complexos e fórmulas matemáticas. Os modelos de linguagem visual (VLMs) para parsing de documentos de ponta a ponta simplificam essa arquitetura tratando a imagem do documento como uma modalidade de entrada única, preservando a ordem natural de leitura e o conteúdo estruturado. Modelos de código aberto treinados especificamente para esse fim — como o olmOCR-2, o eficiente em tokens DeepSeek-OCR (3B) e o ultracompacto PaddleOCR-VL — têm gerado resultados altamente eficientes. Embora os VLMs reduzam a complexidade arquitetural ao substituir pipelines de múltiplos estágios, sua natureza generativa os torna propensos a alucinações. Casos de uso com baixa tolerância a erros ainda podem exigir uma abordagem híbrida ou OCR determinístico. Times que lidam com ingestão de grandes volumes de documentos devem avaliar essas abordagens unificadas para determinar se elas podem substituir pipelines legados complexos, mantendo a precisão e reduzindo o overhead de manutenção a longo prazo.

34. Inchaço de instruções para agentes

Cautela

Arquivos de contexto como o AGENTS.md e CLAUDE.md tendem a se acumular ao longo do tempo à medida que os times adicionam visões gerais da base de código, explicações arquiteturais, convenções e regras. Embora cada adição seja útil isoladamente, isso frequentemente leva ao inchaço de instruções para agentes. As instruções se tornam longas e às vezes conflitam entre si. Os modelos tendem a prestar menos atenção ao conteúdo enterrado no meio de contextos longos, então orientações profundas em um longo histórico de conversas podem ser perdidas. À medida que as instruções crescem, aumenta a probabilidade de que regras importantes sejam ignoradas. Também vemos muitos times usando IA para gerar arquivos AGENTS.md, mas pesquisas sugerem que versões escritas à mão são frequentemente mais eficazes do que as geradas por LLMs. Ao usar ferramentas baseadas em agentes, seja intencional e seletivo com as instruções. Adicione-as conforme a necessidade e refine-as continuamente em direção a um conjunto mínimo e coerente.

35. Shadow IT acelerada por IA

Cautela

A IA continua a diminuir as barreiras para que pessoas que não programam construam sistemas complexos. Embora isso permita a experimentação e a validação antecipada de requisitos, também introduz o risco da shadow IT acelerada por IA. Além de plataformas de workflow no-code que integram APIs de IA (por exemplo, OpenAI ou Anthropic), mais ferramentas agênticas estão se tornando acessíveis a não programadores, como o Claude Cowork. Quando a planilha que silenciosamente administra o negócio evolui para workflows agênticos customizados que carecem de governança, isso introduz riscos de segurança significativos e uma proliferação de soluções concorrentes para problemas semelhantes. Distinguir entre workflows descartáveis e pontuais e processos críticos que exigem uma implementação durável e pronta para produção é fundamental para equilibrar a experimentação com o controle. As organizações devem priorizar a governança como parte de sua estratégia de adoção de IA, facilitando a experimentação dentro de ambientes controlados. Sandboxes internas adequadamente instrumentadas oferecem às pessoas não

programadoras um espaço para implantar protótipos com rastreamento de uso. Combiná-las com um catálogo compartilhado de workflows existentes ajuda as equipes a descobrir o que já foi construído antes de duplicar esforços. Os workflows que ganharem tração podem então sinalizar onde investir em aplicações mais robustas e em nível de produção.

36. Dívida cognitiva da base de código

Cautela

A dívida cognitiva da base de código é a lacuna crescente entre a implementação de um sistema e o entendimento compartilhado de um time sobre como e por que ele funciona. À medida que a IA aumenta a velocidade das mudanças, especialmente com múltiplos contribuidores ou enxames de agentes de programação, os times podem perder o controle da intenção de design e do acoplamento oculto. Isso, combinado com a crescente dívida técnica, cria um ciclo de retroalimentação que torna os sistemas progressivamente mais difíceis de compreender. Um entendimento mais fraco do sistema também reduz a capacidade das pessoas desenvolvedoras de orientar a IA de forma eficaz, tornando mais difícil antecipar edge cases e afastar os agentes de armadilhas arquiteturais. Se não for gerenciado, os times chegam a um ponto de inflexão onde pequenas mudanças desencadeiam falhas inesperadas, correções introduzem regressões e esforços de refatoração aumentam o risco em vez de reduzi-lo. Os times devem evitar a complacência com código gerado por IA e adotar contramedidas explícitas: sensores de feedback para agentes de programação, rastreamento da carga cognitiva do time e funções de aptidão arquitetural para garantir continuamente o cumprimento de restrições essenciais à medida que a IA acelera a produção.

37. Enxames de agentes de programação

Cautela

Enquanto um time de agentes de programação é um grupo pequeno e intencional, um enxame de agentes de programação aplica dezenas a centenas de agentes a um problema, com a IA determinando a composição e o tamanho dinamicamente. Projetos como Gas Town e Ruflo (anteriormente Claude Flow) são bons exemplos dessa abordagem. Padrões emergentes para implementações de enxames estão surgindo: separação hierárquica de papéis (orquestradores, supervisores e trabalhadores efêmeros), um registro de trabalho durável que ajuda os agentes a dividir e coordenar o trabalho (o Gas Town usa beads para isso) e um mecanismo de integração para lidar com conflitos de trabalhos paralelos. Dois experimentos com enxames chamaram atenção especial: a geração de compilador C da Anthropic e o experimento de escalonamento de agentes do Cursor, que criou um navegador em uma semana. Vale notar que ambos os times escolheram casos de uso que podiam depender de especificações detalhadas existentes e, no caso do compilador C, de suítes de testes abrangentes que fornecem feedback claro e mensurável. Essas condições não são representativas do desenvolvimento típico de produtos, onde os requisitos são menos definidos e a verificação é mais difícil. No entanto, esses experimentos contribuem para padrões emergentes que tornam enxames de longa duração tecnicamente viáveis. Eles continuam caros e ainda estão longe de serem maduros, e é por isso que aconselhamos cautela ao adotar essa técnica.

38. Taxa de transferência de programação como medida de produtividade

Cautela

Assistentes de programação baseados em IA estão entregando ganhos reais de produtividade e tornando-se rapidamente ferramentas padrão no desenvolvimento. No entanto, estamos vendo cada vez mais organizações medirem o sucesso usando indicadores superficiais, como linhas de código geradas ou o número de pull requests (PRs). Quando essas métricas de vazão de código (*coding*

throughput) são usadas isoladamente, elas podem moldar negativamente o comportamento das equipes. O resultado é frequentemente uma enxurrada de código desalinhado que atrasa revisões, prejudica o fluxo de entrega e introduz riscos de segurança. Os tempos de ciclo aumentam à medida que engenheiros abrem PRs repletos de sugestões da IA sem a devida revisão, gerando um vaivém constante com os revisores. Essas métricas falham em capturar o esforço residual necessário para adaptar o código gerado por IA à arquitetura, às convenções e aos padrões de cada equipe. Existem indicadores mais significativos, como a taxa de aceitação de primeira passagem — a frequência com que a saída da IA pode ser utilizada com o mínimo de retrabalho. Medir isso expõe o esforço oculto e torna a melhoria acionável: os times podem refinar prompts, melhorar os documentos de contexto e fortalecer as discussões de design para aumentar progressivamente a aceitação. Isso cria um ciclo virtuoso no qual a IA exige menos correções. A aceitação de primeira passagem também se conecta naturalmente às Métricas DORA: baixas taxas de aceitação tendem a elevar a taxa de falha em mudanças, enquanto ciclos de iteração repetidos estendem o lead time. À medida que assistentes de IA se tornam onipresentes, as organizações devem mudar o foco da vazão de código bruta para métricas que reflitam o impacto real e os resultados da entrega.

39. Ignorar a durabilidade em workflows de agentes

Cautela

Ignorar a durabilidade em fluxos de trabalho de agentes é um antipadrão que temos visto em muitas equipes, resultando em sistemas que funcionam no desenvolvimento, mas falham em produção. Os desafios enfrentados por sistemas distribuídos são ainda mais pronunciados ao construir soluções com agentes. Uma mentalidade que espera falhas e se recupera com resiliência supera uma abordagem reativa. Chamadas de LLM e de ferramentas podem falhar devido a interrupções de rede e travamentos de servidor, interrompendo o progresso de um agente e levando a uma experiência ruim para a pessoa usuária e ao aumento dos custos operacionais. Alguns sistemas podem tolerar isso quando as tarefas são de curta duração, mas workflows complexos que rodam por dias ou semanas exigem durabilidade. Felizmente, a execução durável está sendo integrada a frameworks de agentes, como LangGraph e Pydantic AI. Ela fornece persistência stateful do progresso e das chamadas de ferramentas, permitindo que os agentes retomem as tarefas após falhas. Para workflows que envolvem um humano na supervisão, a execução durável pode suspender o progresso enquanto aguarda a entrada de dados. Plataformas de computação durável como Temporal, Restate e Golem também fornecem suporte para agentes. A observabilidade integrada da execução de ferramentas e o rastreamento de decisões facilitam o debugging e melhoram o entendimento dos sistemas em produção. Os times devem começar com o suporte nativo à execução durável em seu framework de agentes e recorrer a plataformas independentes à medida que os workflows se tornam mais críticos ou complexos.

40. MCP por padrão

Cautela

À medida que o Model Context Protocol (MCP) ganha adoção, estamos vendo times e fornecedores recorrerem a ele como a camada de integração padrão entre agentes de IA e sistemas externos, mesmo quando existem alternativas mais simples. Nós alertamos contra o uso do MCP por padrão. O MCP agrega valor real para contratos de ferramentas estruturados, fronteiras de autenticação baseadas em OAuth e acesso governado em ambientes multitenant. Ele também introduz o que Justin Poehnelt chama de "custo de abstração": cada camada de protocolo entre um agente e uma API perde fidelidade e, para APIs complexas, essas perdas se acumulam. Na prática, uma CLI bem projetada com um `--help` bem estruturado, respostas JSON estruturadas e tratamento de erros previsível

frequentemente dá aos agentes tudo o que eles precisam sem a sobrecarga do protocolo. Como Simon Willison observa, "quase tudo o que eu conseguiria fazer com MCP pode ser resolvido por uma ferramenta de CLI." Isso não significa rejeitar o MCP. Os times devem evitar adotá-lo por padrão e primeiro se perguntar se o sistema realmente exige interoperabilidade em nível de protocolo. O MCP faz sentido quando seus benefícios de governança e integração superam a complexidade adicional e a potencial perda de fidelidade.

41. Ambientes de desenvolvimento com streaming de pixels

Cautela

Os ambientes de desenvolvimento com streaming de pixels usam desktops remotos ou workstations no estilo VDI para o desenvolvimento de software, com edição, compilações e debugging realizados por meio de um desktop transmitido via streaming, em vez de uma máquina local ou um ambiente remoto centrado em código. Continuamos a ver organizações adotando-os para atender a metas de segurança, padronização e onboarding, especialmente para times offshore e programas de lift-and-shift para a nuvem. Na prática, no entanto, o trade-off costuma ser ruim: latência, input lag e responsividade inconsistente da tela criam um atrito cognitivo constante que atrasa a entrega e torna o trabalho diário de desenvolvimento mais cansativo. Diferente dos ambientes de desenvolvimento na nuvem, como o Google Cloud Workstations ou ferramentas como o Coder e o VS Code Remote Development, que aproximam o processamento do código sem transmitir todo o desktop, os ambientes com streaming de pixels priorizam o controle centralizado em detrimento do fluxo da pessoa desenvolvedora e frequentemente são impostas com pouca participação das pessoas engenheiras que as utilizam. Desaconselhamos ambientes de desenvolvimento com streaming de pixels como uma escolha padrão para a entrega de software, a menos que uma exigência regulatória ou de segurança muito forte supere claramente o custo de produtividade.

Platforms

Adote

—

Experimente

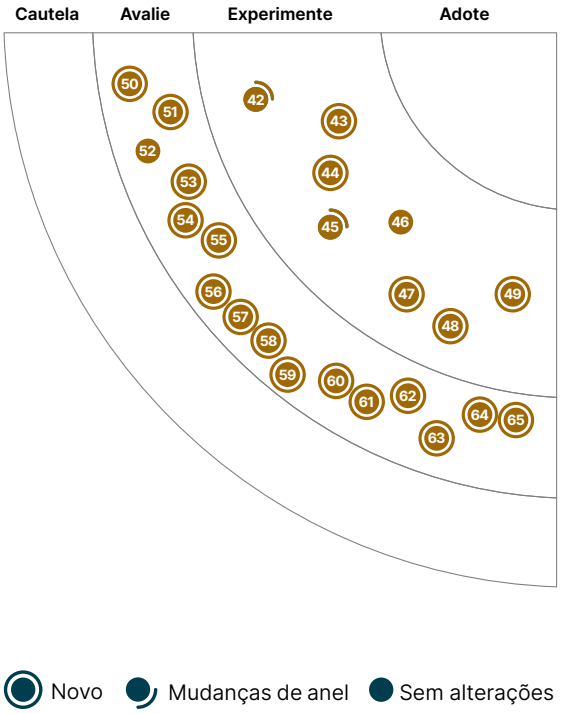
42. AG-UI Protocol
43. Apache APISIX
44. AWS Bedrock AgentCore
45. Graphiti
46. Langfuse
47. Port
48. Replit
49. SigNoz

Avalie

50. Agent Trace
51. ClickStack
52. Coder
53. Databricks Agent Bricks
54. DuckLake
55. FalkorDB
56. Google Dialogflow CX
57. MCP Apps
58. Monarch
59. Neutree
60. OptScale
61. Rhesis
62. RunPod
63. Sprites
64. torchforge
65. torchtitan

Cautela

—



42. AG-UI Protocol

Experimente

O AG-UI é um protocolo aberto e uma biblioteca projetados para padronizar a comunicação entre interfaces de usuário ricas e agentes de IA no backend. Historicamente, construir UIs baseadas em agentes exigia uma infraestrutura sob medida para colaboração bidirecional e stateful. O AG-UI aborda isso fornecendo uma arquitetura consistente e orientada a eventos — com suporte a mecanismos de transporte como server-sent events (SSE) e WebSockets — para transmitir, em streaming, etapas do raciocínio, sincronizar o estado e renderizar componentes dinâmicos de UI. No entanto, o cenário arquitetural para interfaces de agentes está mudando rapidamente. O AG-UI intencionalmente se posiciona fora do MCP, funcionando como uma camada de interface entre o frontend e o backend do agente. Agora estamos vendo uma abordagem diferente emergir, onde aplicações mais recentes baseadas em MCP empacotam HTML e widgets de UI diretamente dentro de servidores MCP ou skills. Como os componentes de UI agora podem ser incorporados e disponibilizados junto com as próprias ferramentas — um padrão relacionado a padrões adjacentes emergentes como o MCP-UI —, a necessidade de uma camada de protocolo de UI separada, como o AG-UI, está sendo questionada. Embora o AG-UI continue sendo uma escolha sólida para desacoplar a UX do frontend da orquestração do backend, os times devem avaliar seu papel à luz da tendência crescente de consolidar a lógica de ferramentas e a UI dentro do ecossistema MCP.

43. Apache APISIX

Experimente

O Apache APISIX é um gateway nativo de nuvem, de alto desempenho e código aberto, que resolve as limitações das soluções legadas baseadas em Nginx. Construído sobre Nginx e LuaJIT via OpenResty, ele utiliza etcd para o armazenamento de configurações, eliminando a latência causada por recarregamentos (*reloads*), o que o torna ideal para microsserviços dinâmicos e arquiteturas *serverless*. Sua principal força é a arquitetura totalmente dinâmica e extensível, que permite às equipes personalizar o gerenciamento de tráfego, a segurança e a observabilidade por meio de APIs e de um ecossistema de plugins multilíngue, incluindo suporte a WASM. Ao suportar a API de Gateway do Kubernetes, o Apache APISIX pode ser utilizado como um gateway para Kubernetes e é um forte candidato para substituir controladores de *ingress* legados do Nginx. Algumas de nossas equipes estão adotando o Apache APISIX e consideram seu desempenho e conjunto de recursos impressionantes.

44. AWS Bedrock AgentCore

Experimente

O AWS Bedrock AgentCore é a plataforma baseada em agentes para construir, executar e operar agentes em escala, com segurança, sem o overhead de gerenciamento de infraestrutura, semelhante ao GCP Vertex AI Agent Builder e ao Azure AI Foundry Agent Service. Embora seja tentador adotar a plataforma como uma caixa preta monolítica, temos visto mais sucesso com uma arquitetura mais refinada e desacoplada: usar o runtime do AgentCore para questões de produção, como isolamento de sessão, segurança e observabilidade, mantendo a lógica de orquestração em frameworks externos como o LangGraph. Essa separação de responsabilidades permite que os times se beneficiem da infraestrutura gerenciada enquanto mantêm a flexibilidade para se adaptar à medida que o cenário de LLMs evolui. Ao focar primeiro no runtime, as organizações podem mover gradualmente cargas de trabalho baseadas em agentes para produção sem ceder o controle de sua lógica central a uma camada de orquestração específica de um fornecedor.

45. Graphiti

Experimente

Estamos movendo o Graphiti para Avaliação (Trial), pois este mecanismo de grafo de conhecimento temporal de código aberto da Zep tem demonstrado viabilidade em produção no endereçamento do problema de memória de LLMs. Enquanto banco de dados vetoriais planos em pipelines de RAG falham em rastrear como os fatos mudam ao longo do tempo, o Graphiti ingere episódios discretos e mantém janelas de validade bitemporal nas arestas do grafo, de modo que fatos desatualizados são invalidados em vez de sobrescritos. Diferente do GraphRAG, que é orientado a processamento em lote, ele atualiza o grafo de forma incremental e entrega recuperação em menos de um segundo via recuperação híbrida que combina busca semântica, BM25 e travessia de grafos, sem chamadas de LLM no momento da consulta. Dois fatores impulsionaram esse movimento: benchmarks revisados por pares relatando melhorias de 18,5% na precisão e reduções de 90% na latência, além do lançamento de um servidor MCP de primeira classe que permite que agentes compatíveis com o Model Context Protocol anexem memória temporal persistente com mínimo esforço de integração. A forte adoção da comunidade sinaliza ainda mais a prontidão para produção. Estamos usando o Graphiti para construir agentes cientes de contexto com grafos de conhecimento stateful e com consciência temporal, e recomendamos avaliá-lo para aplicações baseadas em agentes. O Neo4j é o backend principal, com o FalkorDB como uma alternativa mais leve. Os times também devem considerar os custos de extração de LLM por gravação e fixar dependências (pin dependencies) devido ao seu status de lançamento pré-1.0.

46. Langfuse

Experimente

O Langfuse é uma plataforma de engenharia de LLM de código aberto que abrange observabilidade, gerenciamento de prompts, avaliações e gerenciamento de conjuntos de dados. O projeto amadureceu significativamente desde a última vez que o avaliamos. A arquitetura v3 introduz o ClickHouse, Redis e S3 como componentes de backend, tornando-o mais escalável, mas também mais complexo de hospedar por conta própria. Tanto os SDKs para Python quanto para TypeScript agora são baseados nativamente em OpenTelemetry, fazendo do Langfuse uma escolha natural para times que já usam observabilidade baseada em OTEL. Novos recursos, como o SDK de execução de experimentos e o suporte a saída estruturada para experimentos de prompt, movem o Langfuse além do tracing puro, em direção a fluxos de trabalho sistemáticos de avaliação. Isso faz com que valha a pena considerá-lo em um cenário cada vez mais concorrido que inclui Arize Phoenix, Helicone e LangSmith. Equipes que desenvolvem principalmente com base em Pydantic AI também podem considerar o Pydantic Logfire, que adota uma abordagem mais ampla como uma plataforma de observabilidade OTEL full-stack, e não uma suíte de ferramentas específica para LLMs. O Langfuse permanece em Avalie, pois é uma opção sólida para times que precisam de tracing integrado, avaliações e gerenciamento de prompts em uma única plataforma que pode ser hospedada por conta própria. No entanto, os times devem avaliar se o compromisso com a infraestrutura se justifica para a sua escala e se uma ferramenta mais focada, como o Helicone pode ser suficiente caso a necessidade principal seja visibilidade de custos e latência na camada do modelo.

47. Port

Experimente

O Port é um portal interno comercial para pessoas desenvolvedoras, projetado para melhorar a experiência da pessoa desenvolvedora centralizando ativos de software, automatizando workflows e impondo padrões de engenharia, dando aos times de plataforma uma única fonte de verdade para

workflows de autosserviço. Vemos que isso importa cada vez mais à medida que as organizações buscam padronizar os workflows de engenharia enquanto expõem templates, APIs, automações e agentes de uma forma que as pessoas desenvolvedoras possam realmente usar, inclusive diretamente na IDE através das camadas de API e MCP do Port, e não apenas por meio de um portal independente. Em nossa experiência, o Port funciona bem para organizações que desejam capacidades de portal produzidas sem investir pesadamente em engenharia de plataforma. Em engajamentos com clientes, ele tem suportado milhares de pessoas desenvolvedoras ao mesmo tempo em que permite que times de plataforma relativamente pequenos entreguem um autosserviço eficaz rapidamente. Acreditamos que vale a pena avaliar o Port para organizações que precisam de capacidades de portal interno rapidamente e podem aceitar as restrições de uma plataforma comercial e a dependência de fornecedor.

48. Replit

Experimente

O Replit é uma plataforma de desenvolvimento colaborativo nativa de nuvem que fornece ambientes de desenvolvimento instantâneos, programação em tempo real e assistência de IA integrada diretamente no navegador. Ele combina um editor, runtime, implantação e workflows de programação de IA em uma única plataforma unificada, o que permite que as pessoas desenvolvedoras comecem a programar imediatamente sem nenhuma configuração local. Descobrimos que essa IDE colaborativa baseada em IA é realmente útil para tornar a integração inicial mais fácil, tornando-a uma ótima opção para a prototipagem em time. Também a consideramos muito eficaz para sessões de treinamento, compartilhamento de conhecimento e bootcamps. Embora alguns possam ver o Replit como um lugar para projetos de hobby assistidos por IA, nós achamos que ele se destaca porque o ambiente é poderoso o suficiente para competir com IDEs locais tradicionais, tornando a iteração e a colaboração muito mais fáceis.

49. SigNoz

Experimente

O SigNoz é uma plataforma de observabilidade de código aberto e nativa de OpenTelemetry que oferece suporte unificado a logs, métricas e traces. Atende às necessidades de APM e instrumentação de microsserviços modernos e arquiteturas distribuídas, evitando lock-in de fornecedor. Ao usar ClickHouse como banco de dados colunar subjacente, o SigNoz oferece armazenamento escalável, de alto desempenho e custo-efetivo com consultas rápidas, posicionando-se como uma alternativa forte self-hosted a plataformas como Datadog. Suporta consultas flexíveis via PromQL e SQL do ClickHouse, além de alertas em vários canais de notificação. Na prática, temos visto o SigNoz reduzir o consumo de recursos de infraestrutura e os custos gerais de observabilidade sem comprometer o desempenho. Embora exista um serviço de nuvem gerenciado, imagens Docker e Helm charts prontos para uso tornam-no uma opção prática para organizações que preferem manter o controle sobre dados e infraestrutura.

50. Agent Trace

Avalie

O Agent Trace é uma especificação aberta proposta pelo Cursor que busca padronizar a atribuição de código de IA. À medida que a adoção de agentes de programação aumenta, entender quem modificou o código agora vai além das pessoas desenvolvedoras humanas para incluir alterações geradas por IA. Estamos vendo um interesse inicial de times que precisam de melhor rastreabilidade em torno dessas mudanças. Ferramentas existentes como o git blame podem mostrar que uma linha

de código foi modificada, mas falham em capturar se essa mudança foi feita por um humano, uma IA ou ambos. O Agent Trace adota uma abordagem neutra em relação a fornecedores para definir como as alterações de código são rastreadas e não é opinativo sobre como esses rastros são armazenados. Ele é compatível com múltiplos sistemas de controle de versão, incluindo Git, Mercurial e Jujutsu. A especificação define tipos de contribuintes como humano, IA, misto e desconhecido, juntamente com um registro de rastro descrevendo a origem de cada contribuição. Há sinais iniciais de adoção, com suporte de ferramentas como Cline e OpenCode, bem como implementações como o Git AI. Times que adotam agentes de programação devem avaliar ferramentas que implementam a especificação Agent Trace para melhorar a atribuição de código.

51. ClickStack

Avalie

O ClickStack é uma plataforma de observabilidade de código aberto e compatível com o OpenTelemetry que unifica logs, traces, métricas e sessões em um único armazenamento de dados de alto desempenho construído sobre o ClickHouse. À medida que a infraestrutura cresce e os custos de observabilidade aumentam, muitos times enfrentam dificuldades com conjuntos de ferramentas de telemetria fragmentados e plataformas comerciais caras. O ClickStack aborda esse desafio aproveitando o armazenamento colunar do ClickHouse para permitir consultas de alta cardinalidade em subsegundos em grandes volumes de dados de telemetria, oferecendo uma base mais simples e de melhor custo-benefício para observabilidade.

52. Coder

Avalie

O Coder apresenta uma boa alternativa aos ambientes de desenvolvimento transmitidos por pixels ao separar onde o código é executado de como as pessoas desenvolvedoras interagem com ele. Em vez de transmitir interfaces completas de desktop, as pessoas desenvolvedoras se conectam a ambientes remotos usando IDEs locais, como o VS Code, ou um navegador, resultando em uma experiência mais responsiva sem comprometer a usabilidade. Nesse modelo, o código é executado em infraestrutura remota e escalável, enquanto os ambientes são definidos e gerenciados como código. Isso permite que os times padronizem as configurações de desenvolvimento e simplifiquem o onboarding de novas pessoas desenvolvedoras. Também facilita o fornecimento de acesso controlado a sistemas internos e simplifica o acesso a agentes de programação de IA pré-aprovados. Nós vemos o Coder como um meio-termo entre o desenvolvimento local e desktops totalmente virtualizados: ele fornece controle centralizado e governança sem as limitações de usabilidade de VDIs transmitidos por pixels. Isso o torna uma boa opção para organizações que exigem ambientes de execução remotos ou controlados, particularmente onde mais recursos computacionais ou acesso seguro é necessário. Como em abordagens semelhantes, os times devem avaliar o overhead operacional e as responsabilidades de segurança que vêm com o gerenciamento desses ambientes.

53. Databricks Agent Bricks

Avalie

À medida que as abordagens baseadas em agentes se tornam mais comuns, estamos vendo as plataformas de dados evoluírem para suportar essas cargas de trabalho nativamente, em vez de como um complemento improvisado. O Databricks Agent Bricks fornece componentes pré-construídos e de otimização automática para padrões comuns de IA, como assistentes de conhecimento e analistas de dados. Ele segue uma abordagem declarativa: as pessoas desenvolvedoras definem o objetivo e os dados subjacentes, enquanto o framework lida com a execução e a otimização. Ao simplificar

o LLMOps e reduzir o esforço necessário para a curadoria de dados, os times podem focar mais nos resultados de negócios do que no boilerplate. Por exemplo, nossos times usaram-no ao lado de agentes personalizados para avaliar e construir soluções complexas de RAG para R&D pré-clínico. Se você já investiu no ecossistema Databricks e está explorando abordagens baseadas em agentes para casos de uso comuns, como chatbots e extração de documentos, considere avaliar o Agent Bricks.

54. DuckLake

Avalie

O DuckLake é um formato integrado de data lake e catálogo que simplifica a arquitetura de Lakehouse usando bancos de dados SQL padrão para gerenciar o catálogo e os metadados. Enquanto formatos abertos de tabelas tradicionais, como Iceberg ou Delta Lake, dependem de estruturas complexas de metadados baseadas em arquivos, o DuckLake armazena os metadados em um banco de dados de catálogo (por exemplo, SQLite, PostgreSQL ou DuckDB) enquanto persiste os dados como arquivos Parquet no disco local ou em armazenamento de objetos compatível com S3. Essa abordagem híbrida melhora a latência de planejamento de consultas e a confiabilidade transacional durante atualizações concorrentes. DuckDB atua como mecanismo de consulta por meio da extensão `ducklake`, oferecendo uma interface SQL familiar para operações DDL e DML padrão. Ele também preserva características de lakehouse, como particionamento, mas omite índices e chaves primárias e estrangeiras. Com suporte para time travel, evolução de esquema e conformidade ACID, o DuckLake oferece uma opção de baixa complexidade para times que buscam uma stack analítica autônoma. Embora ainda esteja em estágio inicial de maturidade, o DuckLake é uma alternativa leve e promissora às arquiteturas de lakehouse tradicionais. Ele evita o overhead operacional associado a ecossistemas baseados em Spark ou Trino, o que o torna uma boa opção para ambientes de dados mais enxutos.

55. FalkorDB

Avalie

O FalkorDB é um banco de dados de grafos baseado em Redis que suporta Cypher e atende times que desejam recursos de grafos sem adotar uma plataforma de grafos pesada. Nós o vemos como uma opção prática para organizações que constroem cargas de trabalho de aplicações e IA ricas em relacionamentos, onde o baixo atrito operacional é importante e onde um serviço de grafos baseado em servidor é preferível ao armazenamento embarcado (embedded storage). Estamos colocando-o em Análise (Assess) porque a arquitetura é promissora e o modelo para as pessoas desenvolvedoras é acessível, mas os times devem validar o comportamento em produção em relação ao escalonamento, ferramentas operacionais e maturidade do ecossistema a longo prazo antes de se comprometerem com uma adoção ampla.

56. Google Dialogflow CX

Avalie

O Google Dialogflow CX é a plataforma de IA conversacional gerenciada do Google Cloud, combinando uma máquina de estados baseada em grafos de Fluxos e Páginas com capacidades generativas impulsionadas pelo Vertex AI Gemini. Nós acompanhamos anteriormente seu antecessor, o Dialogflow, no Radar. O CX representa um redesenho significativo que ganhou tração após o Google integrar os modelos Vertex AI Gemini em 2024, introduzindo Generative Playbooks para agentes orientados a instruções e Data Store RAG para fundamentar respostas em conteúdo indexado. Nós o utilizamos para construir um agente de descoberta de dados em linguagem natural, escolhendo-o em vez de uma abordagem de SDK customizado devido ao seu ambiente low-code e aos Generative Playbooks. Nós os configuramos com engenharia de prompt com poucos exemplos para traduzir

consultas em linguagem natural para SQL. Times no Google Cloud que constroem interfaces de linguagem natural sobre dados internos estruturados descobrirão que o Dialogflow CX acelera a entrega em comparação com uma stack de agentes customizada. No entanto, a plataforma não possui um plano gratuito; sua profunda dependência do Google Cloud introduz um "vendor lock-in" (aprisionamento tecnológico) significativo, e as equipes devem planejar o esforço necessário para a engenharia de contexto.

57. MCP Apps

Avalie

O MCP Apps é a primeira extensão oficial para o Model Context Protocol, permitindo que servidores MCP retornem interfaces HTML interativas como dashboards, formulários e visualizações que são renderizadas diretamente na conversa. Desenvolvida em conjunto pela Anthropic, OpenAI e pessoas contribuidoras de código aberto, a extensão padroniza um esquema de recursos `ui://` onde as ferramentas declaram templates de UI renderizados em iframes em sandbox, com degradação suave para texto quando o host não possui suporte a UI. Diferentemente do AG-UI, que opera como uma camada de biblioteca separada, o MCP Apps empacota a UI diretamente dentro dos servidores MCP. O design bidirecional permite que os modelos observem as ações das pessoas usuárias, enquanto a interface lida com dados em tempo real e manipulação direta que o texto não consegue fazer. Clientes como Claude, ChatGPT, VS Code e Goose já vêm com suporte. Times que exploram interações mais ricas com agentes devem avaliar se a complexidade adicional em relação às respostas em texto simples se justifica para o seu caso de uso.

58. Monarch

Avalie

O Monarch é um framework de programação distribuída de código aberto que traz a simplicidade das cargas de trabalho do PyTorch de uma única máquina para grandes clusters de GPU. Ele fornece uma API Python para criar processos remotos e atores, agrupando-os em coleções chamadas "meshes" que suportam mensagens de transmissão. Ele também oferece tolerância a falhas por meio de árvores de supervisão, onde as falhas se propagam acima de uma hierarquia para permitir um tratamento de erros limpo e uma recuperação refinada. Recursos adicionais incluem suporte para transferências RDMA ponto a ponto para movimentação eficiente de memória de CPU e GPU, e uma abstração de tensor distribuído que permite que atores trabalhem com tensores fragmentados entre processos enquanto mantêm um modelo de programação imperativo. O Monarch é construído sobre um backend Rust de alto desempenho. Embora ainda esteja nos estágios iniciais de desenvolvimento, a sua abstração — fazendo com que tensores distribuídos se comportem como locais — é poderosa e pode reduzir bastante a complexidade do treinamento de IA distribuídas de larga escala.

59. Neutree

Avalie

O Neutree é uma plataforma de código aberto para gerenciar e servir LLMs em infraestrutura privada, posicionando-se como uma camada de modelo como serviço (model-as-a-service) para IA corporativa. Ele fornece um plano de controle unificado para gerenciamento do ciclo de vida do modelo, serviço de inferência e agendamento de computação em hardwares heterogêneos, como aceleradores NVIDIA, AMD e Intel. À medida que as organizações se afastam de APIs hospedadas em direção a implantações auto-hospedadas e governadas, o Neutree aborda uma lacuna clara:

operar cargas de trabalho de LLM com capacidades de nível corporativo, como multitenant, controle de acesso, contabilização de uso e abstração de infraestrutura. Ao separar o serviço de modelo da lógica da aplicação, ele permite que os times implantem, escalem e roteiem modelos entre ambientes — incluindo bare metal, VMs (máquinas virtuais) e contêineres — sem um forte acoplamento a um provedor de nuvem específico. No entanto, o Neutree ainda é relativamente novo, e os times devem abordar a adoção com cautela. Seu ecossistema, maturidade operacional e capacidades de integração ainda estão evoluindo, quando comparados a plataformas de ML (machine learning) mais estabelecidas. Embora promissor, ele é mais adequado para times dispostos a investir na avaliação e na formação de infraestruturas emergentes de IA corporativa.

60. OptScale

Avalie

O OptScale é uma plataforma FinOps multi-nuvem de código aberto com suporte para cargas de trabalho pesadas de IA/ML, onde os custos de GPU e experimentação podem disparar rapidamente. Ele ingere dados de faturamento e uso de APIs de nuvem, combinando visibilidade de custos, recomendações de otimização, rastreamento de orçamento e detecção de anomalias em um único sistema com alertas baseados em políticas alinhadas aos times ou estruturas de negócios. Em comparação com o OpenCost, o OptScale cobre casos de uso de FinOps não-Kubernetes mais amplos, enquanto ainda fornece análise no nível do Kubernetes. Ele também oferece mais controle e menos vendor lock-in do que suítes corporativas como IBM Cloudability, CloudZero, CloudHealth, IBM Kubecost e Flexera One. A contrapartida é um overhead operacional mais alto, com preocupações em torno da complexidade de implantação, edge cases de conectores e higiene de segurança de imagens de contêineres. Os times devem tratar o OptScale como um investimento em capacidade de plataforma em vez de um produto plug-and-play.

61. Rhesis

Avalie

O Rhesis é uma plataforma de testes de código aberto para aplicações de LLM e aplicações baseadas em agentes que permite aos times definirem comportamento esperado em linguagem natural, gerar cenários de teste adversariais e avaliar resultados por meio de uma IU e uma SDK ou API. Ele está se tornando mais relevante à medida que as abordagens de teste tradicionais assumem um comportamento determinístico, enquanto os sistemas de IA falham de maneiras mais sutis, incluindo jailbreaks, interações de múltiplos turnos (multi-turn), violações de políticas e edge cases dependentes de contexto. Em nossa avaliação, o Rhesis é uma plataforma útil para times que precisam de mais do que simples avaliações de prompt. Recursos como o simulador de conversas, testes adversariais, tracing baseado em OpenTelemetry e auto-hospedagem via Docker tornam-no uma maneira prática de trazer times de produto, de domínio e de engenharia para um workflow de testes compartilhado. O principal benefício é a melhoria da validação em pré-produção para sistemas não determinísticos. No entanto, os times devem considerar as concessões comuns neste espaço, incluindo o custo de avaliação, os limites das métricas de LLM-como-juiz e a necessidade de requisitos bem definidos antes que a plataforma entregue valor. Nós acreditamos que vale a pena avaliar o Rhesis para times que constroem sistemas de LLMs ou baseados em agentes que exigem testes colaborativos e repetíveis além das verificações básicas de prompt.

62. RunPod

Avalie

À medida que as organizações experimentam cada vez mais o treinamento e ajuste fino (fine-tuning) de LLMs, provedores de hiperescala como AWS e Google Cloud podem introduzir altos custos e disponibilidade limitada de hardware. O RunPod fornece uma alternativa econômica para cargas de trabalho de IA com uso intensivo de computação. Operando como um marketplace de GPUs distribuído globalmente, ele oferece acesso sob demanda a uma ampla gama de hardwares, desde clusters H100 de nível corporativo até RTX 4090s de nível de consumidor, frequentemente a um custo significativamente menor do que os provedores de nuvem tradicionais. Para times que precisam de infraestrutura flexível e econômica para desenvolver, treinar ou implantar modelos de IA sem compromissos de longo prazo ou bloqueios tecnológicos, o RunPod é uma opção prática que vale a pena avaliar.

63. Sprites

Avalie

Sprites é um ambiente de sandbox stateful da Fly.io projetado para executar agentes de programação de IA em isolamento. Enquanto a maioria dos sandboxes de agentes é efêmera, sendo ativada para uma tarefa e desaparecendo em seguida, o Sprites fornece ambientes Linux persistentes com capacidades ilimitadas de checkpoint e restauração. Isso permite que as pessoas desenvolvedoras tirem snapshots de todo o estado do ambiente — incluindo dependências instaladas, configurações de runtime e alterações no sistema de arquivos — e façam rollback quando um agente sai dos trilhos. Isso vai além do que apenas o Git consegue recuperar, capturando o estado do sistema que o controle de versão não rastreia. À medida que nossos times adotam cada vez mais a execução em sandbox para agentes de programação como um padrão sensato, o Sprites representa um dos extremos do espectro: uma abordagem não efêmera e stateful que troca a simplicidade dos contêineres descartáveis por opções mais ricas de recuperação. Os times que avaliam o uso de sandboxes para agentes devem considerar o Sprites ao lado de alternativas efêmeras, como os Dev Containers, com base em suas necessidades e workflow.

64. torchforge

Avalie

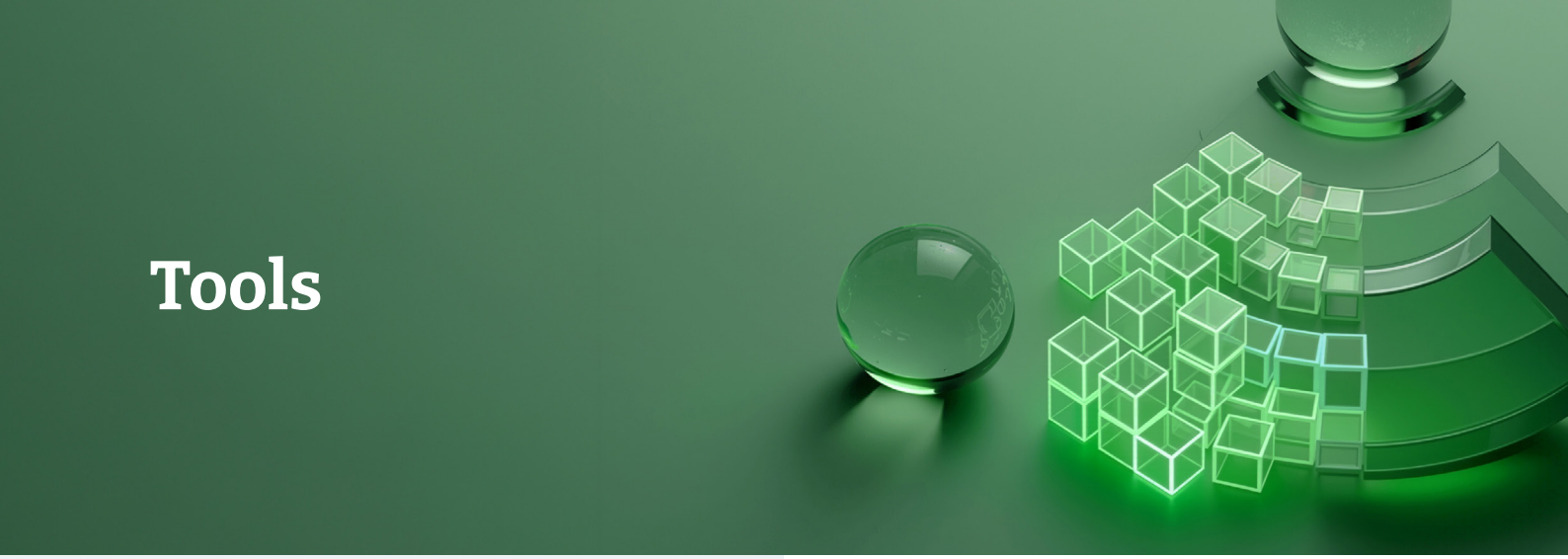
O torchforge é uma biblioteca de aprendizado por reforço nativa do PyTorch projetada para o pós-treinamento em larga escala de modelos de linguagem. Ela fornece uma abstração de mais alto nível que desacopla a lógica algorítmica das questões de infraestrutura, orquestrando componentes como o Monarch para coordenação, vLLM para inferência e torchtitan para treinamento distribuído. Essa abordagem permite que equipes de pesquisa expressem workflows complexos de aprendizado por reforço usando APIs em estilo de pseudocódigo, enquanto distribuem cargas de trabalho por milhares de GPUs sem gerenciar detalhes de baixo nível, como sincronização de recursos, agendamento ou tolerância a falhas. Ao separar o “o que” (concepção do algoritmo) do “como” (execução distribuída), o torchforge simplifica a experimentação e a iteração em sistemas de alinhamento em larga escala. Vemos isso como um passo útil para tornar técnicas avançadas de pós-treinamento mais acessíveis, embora os times devam avaliar sua maturidade e adequação à sua infraestrutura de ML existente.

65. torchtitan

Avalie

O torchtitan é uma plataforma nativa do PyTorch para o pré-treinamento em larga escala de modelos de IA generativa, fornecendo uma implementação de referência limpa e modular para treinamento distribuído de alto desempenho. Ele reúne primitivas distribuídas avançadas em um sistema coeso, suportando paralelismo 4D: paralelismo de dados, de tensor, de pipeline e de contexto. Como o treinamento de modelos na escala do Llama 3.1 405B exige escala e eficiência significativas, o torchtitan oferece uma base prática para construir e operar grandes cargas de trabalho de treinamento. Seu design modular torna mais fácil para os times experimentarem e evoluírem estratégias de paralelismo, mantendo a prontidão para produção. Vemos o torchtitan como um passo útil em direção à padronização do treinamento de modelos em larga escala no ecossistema PyTorch, particularmente para times que constroem sua própria infraestrutura de pré-treinamento.

Tools



Adote

66. Axe-core
67. Claude Code
68. Cursor
69. Kafbat UI
70. mise

Experimente

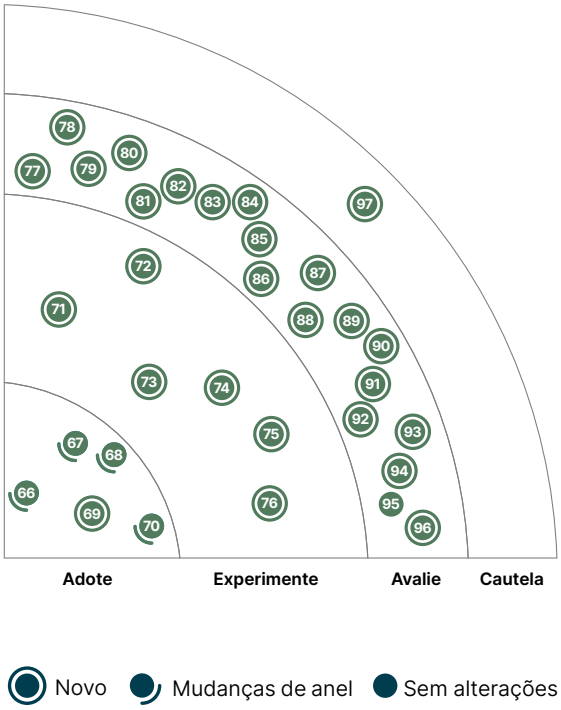
71. cargo-mutants
72. Claude Code plugin marketplace
73. Dev Containers
74. Figma Make
75. OpenAI Codex
76. Typst

Avalie

77. Agent Scan
78. Beads
79. Bloom
80. CDK Terrain
81. CodeScene
82. ConfIT
83. Entire CLI
84. Git-AI
85. Google Antigravity
86. Google Mainframe Assessment Tool
87. OpenCode
88. OpenSpec
89. PageIndex
90. Pencil
91. Pi
92. Qwen 3 TTS
93. SGLang
94. ty
95. Warp
96. WuppieFuzz

Cautela

97. OpenClaw



66. Axe-core

Adote

O Axe-core é uma ferramenta de código aberto para testes de acessibilidade que detecta problemas em sites e outras aplicações baseadas em HTML. Ele verifica as páginas quanto à conformidade com padrões como o WCAG — incluindo os níveis de conformidade A, AA e AAA — e sinaliza as melhores práticas comuns de acessibilidade. Desde que apareceu pela primeira vez no Radar em Trial em 2021, vários de nossos times adotaram o Axe-core com seus clientes. A acessibilidade é cada vez mais um atributo de qualidade obrigatório. Na Europa, por exemplo, regulamentações como a Lei Europeia de Acessibilidade exige que as organizações garantam que seus serviços digitais atendam aos requisitos de acessibilidade. O Axe-core se encaixa bem nos fluxos de trabalho (workflows) de desenvolvimento modernos, permitindo verificações automatizadas em pipelines de CI. Isso ajuda os times a prevenir regressões, manter a conformidade e receber feedbacks antecipados durante o desenvolvimento, garantindo que a acessibilidade faça parte do ciclo de feedback, principalmente à medida que as ferramentas de programação assistidas por IA e baseadas em agentes são adotadas mais amplamente.

67. Claude Code

Adote

O Claude Code da Anthropic é uma ferramenta de IA autônoma para programação para planejar e executar workflows complexos de múltiplas etapas. Estamos trazendo o Claude Code para o Adopt porque times dentro e fora da Thoughtworks agora o utilizam no dia a dia na entrega de software em produção, onde ele é amplamente tratado como um benchmark de capacidade e usabilidade. O cenário das CLI de agentes se expandiu rapidamente com ferramentas como o Codex CLI da OpenAI, o Gemini CLI do Google, o OpenCode e o pi, mas o Claude Code continua sendo a opção preferida para muitos times. Seu uso agora vai além da criação de código para a execução mais ampla de workflows, incluindo especificações, histórias de usuários, configuração, infraestrutura, documentação e processos de negócios definidos em markdown. O Claude Code continua introduzindo recursos que acabam sendo seguidos por outras ferramentas, como skills, subagentes, controle remoto e workflows de times autônomos. Os times que adotam o Claude Code devem combiná-lo com práticas operacionais metódicas. A programação com agentes desloca o esforço da pessoa desenvolvedora da implementação manual para a especificação de intenção, restrições e revisão de limites. Isso pode acelerar a entrega, mas também aumenta o risco de complacência com código gerado por IA, o que pode tornar os sistemas mais difíceis de manter e evoluir tanto para pessoas quanto para agentes. Estamos vendo uma atenção crescente à harness engineering como uma maneira de implementar engenharia de contexto (seleção de contexto baseada em escopo e ciente de tópico) e instruções compartilhadas com curadoria para tornar os workflows baseados em agentes mais confiáveis.

68. Cursor

Adote

O Cursor está entre os agentes de programação mais amplamente adotados que vemos hoje, aparecendo consistentemente ao lado do Claude Code como uma escolha padrão para nossos times de entrega. Ele amadureceu para um ambiente abrangente baseado em agentes com recursos como modo de planejamento, hooks e subagentes. Embora os agentes baseados em terminal continuem populares, muitas de nossas pessoas desenvolvedoras acham que supervisionar um agente dentro de uma IDE fornece uma experiência mais rica para revisar e refinar planos antes da execução. A adoção do Agent Client Protocol reduziu ainda mais as barreiras para a grande base de pessoas

usuárias da Jetbrains, tornando os recursos do Cursor acessíveis dentro dessas IDEs. Valorizamos particularmente a capacidade de inspecionar etapas individuais do agente ou reverter (roll back) para estágios anteriores quando um plano se desvia. Ao alavancar as Skills de agentes, os times podem empacotar instruções reutilizáveis para ajudar a padronizar como os agentes interagem com bases de código complexas. Embora os ganhos de produtividade sejam evidentes, a autonomia do agente ainda requer testes automatizados rigorosos e supervisão humana para capturar regressões sutis.

69. Kafbat UI

Adote

O Kafbat UI é uma interface web gratuita e de código aberto para monitorar e gerenciar clusters do Apache Kafka. Ele tem se mostrado especialmente útil quando as equipes precisam inspecionar payloads que, de outra forma, seriam difíceis de ler durante a depuração no dia a dia. Em nossa experiência, os times frequentemente encontram dificuldades ao depurar mensagens criptografadas, e o suporte do Kafbat UI a SerDes nativos e extensíveis oferece uma maneira prática de aplicar descriptografia ou decodificação personalizada para que as mensagens se tornem legíveis novamente. Em comparação a scripts de depuração pontuais, ele oferece feedback mais rápido e uma melhor experiência operacional para desenvolvedores e equipes de suporte. Recomendamos o Kafbat UI para ambientes com uso intenso de Kafka, onde a inspeção segura de mensagens e a resolução eficiente de problemas devem ser práticas padrão.

70. mise

Adote

Desde nossa última avaliação, o mise evoluiu de uma alternativa de alto desempenho ao asdf para um frontend padrão para o ambiente de desenvolvimento. Estamos movendo-o para Adote porque ele consolida três preocupações fragmentadas — versionamento de ferramentas e linguagens, gerenciamento de variáveis de ambiente e execução de tarefas — em uma única ferramenta de alto desempenho baseada em Rust, configurada por meio de um arquivo declarativo `mise.toml`. O mise é fácil de configurar e funciona bem com pipelines de CI/CD. Ele também adiciona uma camada de segurança da cadeia de suprimentos por meio da integração com o Cosign e o GitHub Artifact Attestations, o que frequentemente falta em outros gerenciadores de versão. Para times que buscam padronizar a configuração de seus ambientes para as pessoas desenvolvedoras, o mise se tornou nosso padrão recomendado. Em ambientes poliglotas com múltiplos microsserviços, isso é especialmente útil quando as bases de código adotam novas versões de linguagem ao mesmo tempo. O melhor de tudo é que o mise também funciona com as ferramentas específicas de linguagem já existentes, para que os times não precisem migrar tudo de uma vez.

71. cargo-mutants

Experimente

O cargo-mutants é uma ferramenta de teste de mutação para Rust que ajuda nossos times a ir além de simples métricas de cobertura de código. Ao injetar automaticamente pequenos bugs intencionais — como trocar operadores ou retornar valores padrão — ele verifica se os testes existentes realmente capturam regressões. Descobrimos que sua abordagem de configuração zero é particularmente eficaz; diferente de ferramentas anteriores, ele não exige alterações na árvore de código-fonte. Para times menos experientes em Rust, ele fornece um ciclo de feedback útil, identificando edge cases esquecidos e melhorando a confiabilidade dos testes de unidade e de integração. Essa ferramenta é uma implementação especializada de testes de mutação (mutation testing), que também estamos experimentando em outros ecossistemas. O custo principal é o aumento do tempo de execução dos

testes, pois cada mutante requer uma compilação incremental. Para gerenciar isso, recomendamos focar em módulos específicos durante o desenvolvimento local ou rodar suítes completas de forma assíncrona na CI. Embora, ocasionalmente, os times precisem filtrar mutantes logicamente equivalentes, o aumento na confiança dos testes supera esse ruído adicional.

72. Claude Code plugin marketplace

Experimente

Anteriormente, compartilhar comandos personalizados, agentes especializados, servidores MCP e skills era um processo manual que dependia de pessoas desenvolvedoras copiarem e colarem instruções do Confluence ou de outras fontes externas. Isso frequentemente levava a desvios de versão, com pessoas do time usando instruções de projeto desatualizadas. Nossos times agora estão usando o Claude Code plugin marketplace para distribuir comandos, prompts e skills compartilhados usando um modelo de distribuição baseado em Git. Ao hospedar marketplaces internos do time no GitHub ou em plataformas semelhantes, as organizações podem distribuir esses artefatos de forma mais segura e consistente. As pessoas desenvolvedoras podem, então, sincronizar os workflows e ferramentas impulsionados por IA diretamente em seus ambientes locais via CLI. Outros agentes de programação, como o Cursor, também suportam um plugin marketplace para times, permitindo uma maneira mais otimizada e governada de compartilhar esses artefatos.

73. Dev Containers

Experimente

Os Dev Containers fornecem uma maneira padronizada de definir ambientes de desenvolvimento em contêineres e reprodutíveis usando um arquivo de configuração `devcontainer.json`. Originalmente projetados para dar aos times configurações de desenvolvimento consistentes, os Dev Containers encontraram um novo e atraente caso de uso como ambientes de execução em sandbox para agentes de programação. Executar um agente de programação de IA dentro de um Dev Container o isola do sistema de arquivos do host, das credenciais e da rede, permitindo que os times concedam amplas permissões aos agentes sem colocar a máquina host em risco. A especificação aberta é suportada nativamente pelo VS Code e por ferramentas baseadas no VS Code, como o Cursor. O DevPod estende o suporte do devcontainer para qualquer editor ou workflow de terminal via SSH. Os Dev Containers adotam uma abordagem efêmera por padrão — ou seja, o contêiner é reconstruído a partir da configuração a cada inicialização —, o que fornece um limite de segurança limpo ao custo de ter que reinstalar ferramentas e dependências. Para times que precisam de estado persistente ou recursos de checkpoint e restauração, alternativas como o Sprites adotam uma abordagem diferente. Os Dev Containers também oferecem benefícios de segurança da cadeia de suprimentos além do sandboxing de agentes. Ao definir o conjunto de ferramentas em uma configuração declarativa, os times reduzem a exposição a pacotes comprometidos e a dependências inesperadas nas máquinas das pessoas desenvolvedoras.

74. Figma Make

Experimente

Anteriormente, nós colocamos no Radar a prototipagem de UI de autoatendimento com GenAI, e essa técnica agora é amplamente adotada pelos times de desenvolvimento, incluindo product managers e designers, para gerar protótipos de alta fidelidade que podem ser testados pelas pessoas usuárias finais. O Figma Make é uma boa opção para construir esses protótipos, pois aproveita componentes

e camadas reais do design de um sistema, fazendo com que os resultados se assemelhem muito às aplicações em produção. O Figma Make usa um modelo de IA sob medida, treinado com base em padrões de design de alta qualidade. Nossos times estão usando-o para criar novos design de telas, aprimorar as telas existentes e construir protótipos compartilháveis para coletar feedback rápido das pessoas usuárias.

75. OpenAI Codex

Experimente

O OpenAI Codex evoluiu para uma ferramenta de programação baseada em agentes independente, disponível via CLI e com um aplicativo dedicado para macOS. Ele foi projetado para a delegação autônoma de tarefas: dado um prompt, ele planeja, implementa e itera em arquivos com o mínimo de intervenção. Nossos times o consideraram eficaz como uma ferramenta de rascunho de alta velocidade, particularmente para tarefas greenfield e trabalho de implementação repetitivo. No entanto, sua tendência de sugerir padrões de biblioteca logicamente sólidos, mas funcionalmente desatualizados, significa que testes automatizados e revisão humana são essenciais. Assim como com outras ferramentas baseadas em agentes neste Radar, o risco de acumular dívida técnica sutil é real e proporcional ao nível de autonomia que os times concedem a ele.

76. Typst

Experimente

O Typst é um sistema de composição tipográfica baseado em marcação posicionado como um sucessor moderno do LaTeX para geração programática de documentos. Ele combina tipografia de alta qualidade com uma sintaxe mais simples e um pipeline de compilação significativamente mais rápido, compilando até mesmo documentos muito grandes em uma fração do tempo exigido pelos conjuntos de ferramentas tradicionais do LaTeX. O Typst oferece mensagens de erro mais claras e recursos de script integrados, como condicionais e loops. Ele também é capaz de carregar dados estruturados de JSON ou CSV, tornando-o muito adequado para geração automatizada de documentos. Nossos times usaram o Typst para gerar extratos e relatórios para clientes de serviços bancários e financeiros, onde os documentos devem ser produzidos em escala com formatação consistente. O compilador de código aberto pode ser auto-hospedado, e seu ecossistema em crescimento inclui pacotes contribuídos pela comunidade. O Typst é mais acessível que o LaTeX, ao mesmo tempo em que entrega uma qualidade tipográfica comparável.

77. Agent Scan

Avalie

O Agent Scan é um scanner de segurança para ecossistemas de agentes que identifica componentes locais, incluindo servidores MCP e skills, e sinaliza riscos como injeção de prompt, “tool poisoning”, “toxic flows”, segredos hardcoded e tratamento inseguro de credenciais. Ele aborda uma lacuna emergente na visibilidade da cadeia de suprimentos de agentes e oferece uma forma prática de inventariar e testar superfícies expostas por agentes, em rápida expansão. No entanto, a adoção deve ser criteriosa. Isso ocorre por vários motivos: as varreduras exigem o compartilhamento de metadados de componentes com as APIs da Snyk, e tanto a qualidade dos sinais gerados quanto as taxas de falsos positivos precisam ser validadas no seu ambiente. É importante que os times confirmem o valor operacional do Agent Scan antes de torná-lo parte de gates obrigatórios no processo de entrega.

78. Beads

Avalie

O Beads é um issue tracker baseado em Git, projetado como uma camada de memória persistente para agentes de codificação. Em vez de depender de planos em Markdown ad hoc, ele oferece aos agentes um grafo de tarefas estruturado com relações de dependência/bloqueio, detecção de tarefas prontas para execução e coordenação compatível com branches para trabalhos de longo prazo em várias sessões. Ele é construído sobre o Dolt, um banco de dados SQL com controle de versão integrado que oferece suporte a branches, merge, diffs e clonagem de tabelas de forma semelhante a um repositório Git. O Beads representa uma nova categoria de ferramentas de memória de projeto e de rastreamento de tarefas nativas de agentes. Outros projetos pioneiros neste espaço incluem o ticket e o tracer. Diferente dos sistemas de tickets tradicionais, como GitHub Issues e Jira, o Beads e ferramentas similares viabilizam novos workflows para coordenar a execução autônoma de múltiplos agentes, incluindo agentes que atribuem tarefas uns aos outros.

79. Bloom

Avalie

O Bloom é uma ferramenta da Anthropic para pessoas pesquisadoras de segurança em IA avaliarem o comportamento de LLMs. Ele investiga comportamentos como a bajulação e a autopreservação. Em comparação com benchmarks estáticos, ele usa uma configuração inicial que define os comportamentos alvo e os parâmetros de avaliação para gerar dinamicamente conversas de teste diversificadas e, em seguida, avaliar os resultados. Essa abordagem para avaliação comportamental automatizada é necessária para acompanhar o ritmo dos lançamentos de modelos, permitindo que times de pesquisa externos conduzam avaliações. O Petri é uma ferramenta complementar que identifica quais comportamentos surgem para um determinado modelo, enquanto o Bloom identifica em quais cenários e com que frequência esses comportamentos ocorrem, formando juntos um conjunto de avaliação mais completo. Uma preocupação é que o Bloom requer um modelo professor (ou avaliador) que avalia um determinado modelo aluno. Modelos professores podem ter pontos cegos e vieses, portanto, usar múltiplos avaliadores pode reduzir o viés nos resultados. Times de pesquisa em segurança de IA devem avaliar o Bloom como um complemento aos benchmarks estáticos para avaliar comportamentos emergentes de modelos.

80. CDK Terrain

Avalie

O CDK Terrain (CDKTN) é um fork mantido pela comunidade do Cloud Development Kit for Terraform (CDKTF), que a HashiCorp descontinuou e arquivou em dezembro de 2025. O CDK Terrain retoma de onde o CDKTF parou, permitindo que os times definam infraestrutura usando TypeScript, Python ou Go e a provisionem por meio do Terraform ou OpenTofu. Para times que já investiram no CDKTF, o CDK Terrain oferece um caminho de migração que preserva o código e os workflows existentes, em vez de forçar uma migração para o HCL ou Pulumi. O projeto tem lançamentos mensais e passou a oferecer suporte de primeira classe ao OpenTofu. No entanto, forks mantidos pela comunidade de projetos abandonados por fornecedores trazem riscos inerentes quanto ao suporte de longo prazo, e a abordagem do CDKTF nunca alcançou ampla adoção. A HashiCorp citou a falta de product-market fit ao descontinuá-lo. Os times que usam CDKTF atualmente devem avaliar o CDK Terrain como uma opção de continuidade, mas também ponderar se este é o momento certo para migrar para uma abordagem com suporte mais amplo.

81. CodeScene

Avalie

Nós colocamos no Radar a análise social de código lá em 2017, mas com o aumento da adoção de agentes de programação, estamos vendo um interesse renovado, particularmente em ferramentas como o CodeScene. O CodeScene é uma ferramenta de análise comportamental de código que identifica a dívida técnica combinando métricas de complexidade de código com o histórico do controle de versão. Diferente da análise estática tradicional, ela destaca "hotspots" para ajudar os times a priorizar a refatoração com base na atividade real de desenvolvimento e no impacto nos negócios. O CodeScene agora fornece orientações para o design de código amigável à IA. Nossos times estão descobrindo que a qualidade do código se tornou ainda mais importante, pois os agentes de programação podem modificar o código muito mais rapidamente do que as pessoas desenvolvedoras humanas. A métrica CodeHealth do CodeScene fornece um guardrail útil ao identificar áreas onde o código é complexo demais para os LLMs refatorarem com segurança sem um alto risco de alucinações. Recomendamos que os times avaliem o CodeScene como um guardrail para a adoção de agentes de programação. Sua métrica CodeHealth destaca alvos seguros de refatoração e sinaliza áreas que exigem remediação antes que os agentes sejam aplicados.

82. ConfIT

Avalie

O ConfIT é uma biblioteca para definir testes de API de integração e de componente de forma declarativa em JSON, em vez de escrever fluxos de teste de forma imperativa no código. Estamos vendo um interesse crescente nessa abordagem, pois grandes suítes de testes frequentemente acumulam boilerplate em torno de clientes HTTP, configuração de requisições e asserções. O desenvolvimento assistido por IA reforça ainda mais essa tendência, tornando as definições de teste estruturadas mais fáceis de gerar e manter do que o código procedural verboso. Com base na experiência em clientes e em nossa avaliação, uma camada declarativa pode reduzir a duplicação entre testes de integração e de componentes, melhorar a legibilidade e tornar a intenção do teste mais fácil de evoluir entre os times. No entanto, o próprio ConfIT parece ter uma adoção limitada da comunidade e um ecossistema pequeno, tornando mais difícil recomendá-lo amplamente apesar desses benefícios. Acreditamos que vale a pena avaliar o ConfIT para times .NET que exploram testes de API orientados a especificações. Os times devem, no entanto, validar sua capacidade de manutenção a longo prazo, adequação ao ecossistema e as concessões operacionais antes de adotá-lo mais amplamente.

83. Entire CLI

Avalie

A Entire CLI se conecta aos workflows do Git para capturar sessões de agentes de programação de IA — incluindo transcrições, prompts, chamadas de ferramentas, arquivos modificados e uso de tokens — como metadados pesquisáveis armazenados em uma branch dedicada do repositório. Ela suporta o Claude Code, Gemini CLI, OpenCode, Cursor, Factory AI Droid e GitHub Copilot CLI. À medida que os agentes de IA se tornam os principais contribuidores das bases de código, os times enfrentam uma lacuna crescente entre o que o Git rastreia e o que realmente acontece durante uma sessão de programação. A Entire CLI resolve isso registrando sessões completas junto com os commits, criando uma trilha de auditoria da atividade do agente sem poluir o histórico da branch principal. Seu sistema de checkpoint também permite uma recuperação prática: os times podem retroceder para um estado funcional, quando um agente sai dos trilhos, e retomar de qualquer

checkpoint. Embora a ferramenta ainda seja muito nova e o ecossistema para rastreabilidade de sessões de agentes ainda esteja se formando, os times com requisitos de conformidade ou auditoria relacionados a código gerado por IA podem achar a captura de sessão nativa do Git uma solução ideal.

84. Git-AI

Avalie

O Git AI é uma extensão do Git de código aberto que rastreia código gerado por IA em seus repositórios, vinculando cada linha escrita por IA ao agente, modelo e prompts que a geraram. O Git AI usa checkpoints e hooks para rastrear mudanças incrementais no código entre o início e o fim de um commit. Cada checkpoint contém um diff entre o estado atual e o checkpoint anterior, marcado como sendo de autoria de IA ou humana. Essa abordagem é mais precisa do que métodos focados em rastrear a autoria de código de IA pela contagem de linhas de código no momento da inserção. O Git AI usa um padrão aberto (open standard) para rastrear código gerado por IA usando o Git Notes. Embora o ecossistema de agentes suportados ainda esteja amadurecendo, acreditamos que vale a pena avaliar o Git AI para times que buscam reter a capacidade de manutenção a longo prazo e a responsabilidade em um fluxo de trabalho baseado em agentes. Tanto humanos quanto agentes de IA podem consultar a intenção original e as decisões arquiteturais por trás de um bloco de código específico via skill /ask, referenciando a sessão arquivada do agente.

85. Google Antigravity

Avalie

O Google Antigravity é um fork independente do VS Code, construído com tecnologia licenciada do Windsurf e lançado em prévia pública junto com o Gemini 3 em novembro de 2025. O Antigravity centraliza a IDE em torno da orquestração de múltiplos agentes: um gerenciador de agentes executa vários agentes em paralelo nas tarefas, um navegador Chromium integrado permite que os agentes interajam diretamente com UIs ativas, e um sistema de skills armazena instruções reutilizáveis de agentes no repositório. O gerenciador de agentes atua mais como um painel de “Mission Control” do que apenas uma barra lateral de chat padrão, mudando fundamentalmente o papel da pessoa desenvolvedora de escrever código linha por linha para orquestrar múltiplos fluxos de trabalho autônomos. As pessoas desenvolvedoras ainda podem entrar no editor para assumir o controle com um humano no ciclo (HITL) quando necessário. O Antigravity se integra ao Google Cloud e ao Firebase por meio do Model Context Protocol e suporta o desenvolvimento de agentes via Agent Development Kit. Estamos colocando o Antigravity em Avalie porque ele continua em prévia pública sem data para disponibilidade geral (GA), e sua postura de segurança e prontidão corporativa ainda estão em evolução. O modelo de execução com múltiplos agentes e o acesso autônomo ao navegador sinalizam para onde as IDEs baseadas em agentes estão indo.

86. Google Mainframe Assessment Tool

Avalie

A Google Mainframe Assessment Tool ajuda as organizações a fazer engenharia reversa de aplicações que rodam em mainframes, analisando um portfólio inteiro ou sistemas individuais. Em sua essência, ela depende de parsers determinísticos de linguagem para mapear fluxos de chamadas e dependências de dados em bases de código, criando uma visão estrutural de como as aplicações interagem. A partir dessa base, os recursos de IA generativa oferecem resumos, documentação, geração de casos de teste e sugestões de modernização. Essa abordagem se alinha a um padrão mais amplo de usar GenAI para entender bases de código legadas, onde insights sólidos sobre o

sistema formam a base para o uso eficaz da IA. Embora a ferramenta ainda não ofereça suporte a todas as principais stacks de tecnologia de mainframe, ela está evoluindo rapidamente. Nossos times a consideraram útil em projetos com clientes focados em descoberta e modernização de aplicações de mainframe.

87. OpenCode

Avalie

O OpenCode rapidamente se tornou um dos agentes de programação de código aberto mais proeminentes, com uma experiência robusta e focada no terminal. Um de seus pontos mais fortes é a flexibilidade de modelos: ele suporta modelos de fronteira hospedados, endpoints auto-hospedados e modelos locais. Isso torna o OpenCode atrativo para controle de custos, personalização e ambientes restritos, incluindo configurações isoladas (air-gapped). Isso também significa que as pessoas usuárias precisam ter clareza sobre o licenciamento e os termos do provedor ao usar assinaturas ou APIs. O modelo de extensões do OpenCode é outro grande atrativo, dando suporte a plugins e integrações MCP para workflows, ferramentas e guardrails específicos do time. Muitas pessoas usuárias adicionam o Oh My OpenCode, uma estrutura (harness) opcional, mas popular, que fornece uma configuração mais rígida (opinionated), completa e pronta para uso (batteries-included), com times de agentes coordenados e padrões de orquestração mais ricos.

88. OpenSpec

Avalie

À medida que as capacidades dos agentes de programação de IA evoluem, as pessoas desenvolvedoras enfrentam cada vez mais desafios com a previsibilidade e a capacidade de manutenção quando os requisitos e o contexto vivem apenas em históricos efêmeros de chat. Para resolver isso, estamos vendo o surgimento de ferramentas de desenvolvimento orientado a especificações (SDD). O OpenSpec é um framework SDD de código aberto que introduz uma camada leve de especificação, garantindo que as pessoas desenvolvedoras humanas e os agentes de IA se alinhem sobre o que construir antes que o código seja gerado. O que diferencia o OpenSpec é o seu workflow fluido e mínimo, que muitas vezes é reduzido a três etapas: propor → aplicar → arquivar. Muitos frameworks SDD (por exemplo, GitHub Spec Kit) ou workflows de Skills de agentes (por exemplo, Superpowers) são mais adequados para projetos greenfield do que brownfield. Gostamos particularmente do foco do OpenSpec nos deltas de especificação em vez de definir uma especificação completa antecipadamente, o que o torna muito adequado para sistemas existentes. Diferente de alternativas mais pesadas que impõem workflows mais rígidos (como o BMAD) ou que exigem integrações de IDE específicas de fornecedores (como o Kiro), o OpenSpec é iterativo e agnóstico em relação a ferramentas. Para times que buscam introduzir estrutura e previsibilidade no desenvolvimento assistido por IA sem adotar um processo pesado, o OpenSpec é um framework amigável para pessoas desenvolvedoras que vale a pena avaliar. Enquanto isso, à medida que os modelos e os agentes de programação continuam a se tornar mais poderosos, também recomendamos que os times continuem a monitorar e revisitar as capacidades nativas e a reavaliar a necessidade de ferramentas de SDD.

89. PageIndex

Avalie

O PageIndex é uma ferramenta que constrói um índice hierárquico de um documento para pipelines de RAG baseados em raciocínio e sem vetores, em vez de depender da recuperação tradicional baseada em embeddings. Em vez de dividir um documento em vetores, o que pode perder informações

estruturais e fornecer visibilidade limitada sobre por que os resultados foram recuperados, o PageIndex constrói um índice de sumário que um LLM percorre passo a passo para recuperar o conteúdo relevante. Isso produz um rastro explícito de raciocínio que explica por que uma seção específica foi selecionada, semelhante a como um ser humano escaneia os títulos e se aprofunda em seções específicas. Alguns de nossos times descobriram que essa abordagem funciona bem para documentos em que o significado depende fortemente da estrutura e não da semântica, como relatórios financeiros com dados numéricos, documentos legais com artigos de referência cruzada e documentos clínicos ou científicos complexos. No entanto, essa abordagem traz concessões. Por exemplo, como a inferência do LLM faz parte do processo de recuperação, ela pode introduzir latência e custos significativos, especialmente para documentos grandes.

90. Pencil

Avalie

O Pencil é uma ferramenta de canvas de design que se integra com IDEs e agentes de programação, como o Cursor e o Claude Code. De modo diferente do Figma, que atualmente oferece apenas acesso de leitura, o Pencil executa um servidor MCP local bidirecional que dá acesso tanto de leitura quanto de escrita para manipular o canvas diretamente. Ele também fornece capacidades de design-to-code (design para código) de forma semelhante a ferramentas como o Figma Make e o Builder.io, mas adota uma abordagem mais centrada nas pessoas desenvolvedoras, com arquivos de design armazenados no repositório em um formato JSON aberto chamado .pen, tornando os assets de design versionáveis junto com o código. Essa abordagem pode ajudar a preencher a lacuna na transição de design para desenvolvimento ao se integrar com ferramentas familiares para as pessoas desenvolvedoras. Para design systems complexos e de grande escala, o Figma continua sendo o padrão para colaboração entre diferentes papéis. No entanto, vale a pena considerar o Pencil para times sem profissionais de design dedicados ou para times com pessoas desenvolvedoras que possuem fortes habilidades de design.

91. Pi

Avalie

O pi é um agente de programação de terminal minimalista e de código aberto, escrito em TypeScript. Nós o vemos como algo atraente para pessoas curiosas e experimentadoras, em vez de um padrão corporativo mainstream. O pi é uma estrutura enxuta que é mais personalizável do que um agente completo como o OpenCode. Também é mais fácil de adaptar do que construir um novo agente com um framework baseado em agentes, como ADK, LangGraph ou Mastra. O projeto ainda está em fase inicial e é liderado principalmente por pessoas mantenedoras, apesar da forte tração e de atualizações frequentes. Trate o pi como um building block voltado para pessoas engenheiras, e não como uma plataforma corporativa completa com todos os guardrails e suporte.

92. Qwen 3 TTS

Avalie

O Qwen 3 TTS é um modelo de conversão de texto em fala de código aberto que fecha grande parte da lacuna de qualidade em relação às ofertas comerciais, ao mesmo tempo em que oferece maior controle para as pessoas desenvolvedoras do que muitas APIs pagas. Ele oferece suporte a múltiplos idiomas, pode clonar vozes a partir de amostras curtas (aproximadamente 10 a 15 segundos) e permite o fine-tuning pós-treinamento para vozes específicas de um domínio ou personagem, tornando-o uma opção atraente para times que precisam de fala específica da marca ou controle

on-premises. Ainda é um lançamento recente, e os times devem validar a estabilidade, os controles de segurança, a adequação do licenciamento e a maturidade operacional antes de adotá-lo para workloads de voz críticos em produção.

93. SGLang

Avalie

O SGLang é um framework de serviço de alto desempenho que reduz o overhead de computação da inferência de LLMs por meio de um co-design de sua linguagem de programação de frontend e runtime de backend. Ele introduz o RadixAttention, uma técnica de gerenciamento de memória que armazena em cache e reutiliza de forma agressiva os estados KV (chave-valor) entre prompts. Essa abordagem entrega melhorias significativas de desempenho em relação a motores de serviço padrão, como o vLLM, em cenários com alta sobreposição de prefixos. Para times que constroem agentes autônomos complexos, que dependem de prompts de sistema longos ou usam extensivamente few-shot prompting com exemplos compartilhados, o SGLang pode fornecer ganhos substanciais em latência e eficiência.

94. ty

Avalie

À medida que o Python continua a crescer em popularidade, especialmente no espaço de IA e ciência de dados, ter um sistema de tipos forte torna-se cada vez mais valioso. O ty é um verificador de tipos e servidor de linguagem Python extremamente rápido, escrito em Rust. Ele faz parte do ecossistema Astral, que também inclui ferramentas como o uv e o ruff. O ty fornece feedback rápido e se integra bem a editores comuns, como o Visual Studio Code, entre outros. Em nossa experiência, o uso do ty junto com outras ferramentas da Astral simplifica o desenvolvimento em Python em escala nas grandes organizações. Conforme a programação baseada em agentes se torna mais comum, ter um verificador de tipos determinístico com um ciclo de feedback rápido ajuda a detectar erros cedo e reduz o esforço de revisão de código em erros simples.

95. Warp

Avalie

Desde a última vez que incluímos o Warp no Radar, ele evoluiu muito além de sua descrição de "terminal com recursos de IA". Seus pontos fortes centrais permanecem — saída de comandos baseada em blocos, sugestões impulsionadas por IA e recursos de notebook —, mas o Warp se expandiu para um território tradicionalmente ocupado por IDEs. Ele agora pode renderizar Markdown, exibir uma árvore de arquivos e abrir arquivos diretamente no terminal, suportando um workflow completo de desenvolvimento baseado em agentes em diferentes painéis: um agente de programação como o Claude Code em um, um shell no outro e a visualização de arquivos do workspace em um terceiro painel. Uma vantagem prática que observamos é que o Warp lida com a saída de texto de alta taxa de transferência produzida por agentes de programação modernos melhor do que os terminais tradicionais, onde a velocidade de renderização e a legibilidade podem se tornar gargalos. O Warp também adicionou um assistente de programação integrado, embora isso não tenha sido amplamente avaliado em nossos times. O Warp lançou recentemente o Oz, uma plataforma de orquestração para agentes em nuvem que se integra ao terminal. Este blip foca no próprio terminal. Times que preferem um terminal leve e combinável e desejam trazer suas próprias ferramentas de IA podem achar o Ghostty uma opção melhor; ele adota uma abordagem deliberadamente minimalista em contraste com a filosofia completa (batteries-included) do Warp. O ritmo de novos recursos e as ambições de plataforma mais amplas do Warp tornam a implementação da Avaliação (Trial) prematura até que o produto se estabilize e ganhamos mais experiência de campo com suas capacidades mais recentes.

96. WuppieFuzz

Avalie

O WuppieFuzz é um fuzzer de código aberto para APIs REST que usa uma definição OpenAPI para gerar solicitações válidas, modifica-as para explorar os casos extremos (edge cases) e se baseia no feedback de cobertura do servidor para priorizar entradas que alcançam novos caminhos de execução. Isso é importante porque a maioria dos times ainda depende de testes de integração e de contrato baseados em exemplos, que raramente investigam entradas inesperadas, sequências de solicitação incomuns ou caminhos propensos a falhas, mesmo que as APIs sejam frequentemente a principal superfície de integração de sistemas modernos. Com base em nossa avaliação inicial, o WuppieFuzz parece um complemento promissor para esses testes, porque pode descobrir problemas como exceções não tratadas, lacunas de autorização, vazamentos de dados sensíveis, erros do lado do servidor e falhas de lógica que os testes roteirizados podem deixar passar. Os times ainda precisam avaliar como ele se encaixa na CI, o overhead de runtime que ele introduz e quão úteis são seus resultados na prática. Por essa razão, acreditamos que vale a pena avaliar o WuppieFuzz para times que constroem APIs REST críticas ou expostas externamente.

97. OpenClaw

Cautela

O OpenClaw é um projeto de código aberto na categoria que sua criadora chama de “assistente de IA hiper-pessoal”. As pessoas usuárias hospedam sua própria instância, a mantém continuamente disponível por meio de canais de mensagens como WhatsApp ou iMessage e, em seguida, permitem que ela execute tarefas por meio de ferramentas conectadas. Com memória persistente de conversas, preferências e hábitos, o OpenClaw cria uma experiência pessoal persistente que parece materialmente diferente das interfaces de chat de GenAI ou agentes de programação típicos. O modelo é claramente atraente e já inspirou seguidores como o Claude Cowork. Nós colocamos o OpenClaw em Reflita porque o modelo exige concessões (trade-offs) de segurança substanciais. Quanto mais acesso você concede a ele — ao calendário, e-mail, arquivos e comunicações — mais útil ele se torna, e mais ele concentra permissões exatamente no padrão sobre o qual alertamos em análise de fluxo tóxico para IA. Esse risco não é exclusivo do OpenClaw; ele se aplica a outras implementações do mesmo padrão, incluindo ofertas de fornecedores estabelecidos. Nós publicamos conselhos para times sobre o OpenClaw e ambientes de execução em sandbox, e alternativas como NanoClaw ou ZeroClaw podem reduzir o raio de impacto. No entanto, o padrão de assistente hiper-pessoal por si só continua ávido por permissões e sendo de alto risco.

Languages and Frameworks

Adote

98. Apache Iceberg
99. Declarative Automation Bundles
100. React JS
101. React Native
102. Svelte
103. Typer

Experimente

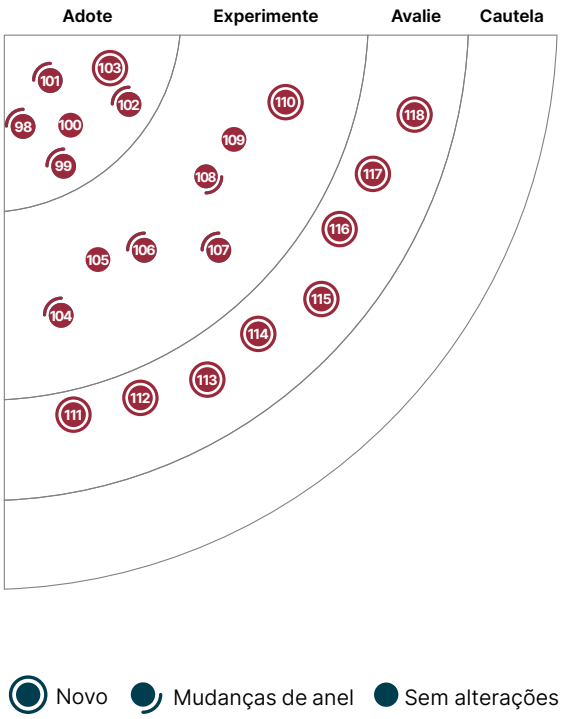
104. Agent Development Kit (ADK)
105. DeepEval
106. Docling
107. LangExtract
108. LangGraph
109. LiteLLM
110. Modern.js

Avalie

111. Agent Lightning
112. GitHub Spec Kit
113. Mastra
114. Pipecat
115. Superpowers
116. TanStack Start
117. TOON (Token-Oriented Object Notation)
118. Unsloth

Cautela

—



Novo Mudanças de anel Sem alterações

98. Apache Iceberg

Adote

O Apache Iceberg é um formato de tabela aberta para conjuntos de dados analíticos em larga escala, que define como os arquivos de dados, metadados e esquemas são organizados em sistemas de armazenamento como o S3. Tendo evoluído significativamente nos últimos anos, ele se tornou um bloco constitutivo (building block) fundamental para arquiteturas lakehouse agnósticas em relação à tecnologia. O Iceberg agora é suportado por todos os principais provedores de plataformas de dados — incluindo AWS (Athena, EMR, Redshift), Snowflake, Databricks e Google BigQuery —, tornando-o uma opção forte para evitar a dependência de fornecedor (vendor lock-in). O que distingue o Iceberg de outros formatos de tabela aberta é sua abertura tanto em recursos quanto em governança, diferente de alternativas cujas capacidades são limitadas ou controladas por um único fornecedor. De uma perspectiva de confiabilidade, o design baseado em snapshots (cópia do estado do sistema) do Iceberg fornece isolamento serializável, escrita simultânea segura por meio de concorrência otimista e com histórico de versões e possibilidade de reversão (rollback). Essas capacidades entregam fortes garantias de estarem corretas enquanto evitam gargalos de desempenho. Embora o Apache Spark continue sendo o motor mais comum usado com o Iceberg, ele também é bem suportado pelo Trino, Flink, DuckDB e outros, tornando-o adequado para uma ampla gama de casos de uso, desde plataformas de dados corporativas até análises locais leves. Em muitos de nossos times, o Iceberg conquistou forte confiança como um formato de dados aberto e estável; recomendado como uma escolha padrão para organizações que constroem plataformas de dados modernas.

99. Declarative Automation Bundles

Adote

O Declarative Automation Bundles (anteriormente conhecido como Databricks Asset Bundles) evoluiu para uma ferramenta primária para trazer práticas de engenharia de software e CI/CD para o ecossistema Databricks. A ferramenta amadureceu significativamente e agora permite que nossos times gerenciem a maioria dos recursos da plataforma como código, incluindo clusters, pipelines de ETL, jobs, modelos de machine learning e dashboards. Usando o comando `bundle plan` do Databricks, os times podem pré-visualizar mudanças e aplicar práticas de implantação repetíveis aos artefatos do Databricks, de forma semelhante a como a infraestrutura é gerenciada com ferramentas como o Terraform. Tratar ativos tradicionalmente mutáveis, como dashboards e pipelines de ML, como código permite que eles sejam controlados por versão, testados e implantados com o mesmo rigor dos microsserviços tradicionais. Com base em nossa experiência em ambientes de produção, o Declarative Automation Bundles se tornou uma abordagem confiável para gerenciar workflows de dados e ML no Databricks. Times que trabalham extensivamente no ecossistema Databricks devem considerar adotá-lo para padronizar as práticas de gerenciamento de infraestrutura.

100. React JS

Adote

O `React` tem sido nossa escolha padrão para o desenvolvimento de UIs em JavaScript desde 2016, mas com o lançamento estável do React Compiler no outubro passado (como parte do React 19), vale a pena revisitá-lo. Há uma série de razões pelas quais esse recurso é notável. Ao lidar com a memoização no momento da compilação (build time), por exemplo, ele torna o uso manual de `useMemo` e `useCallback` amplamente desnecessário, embora o time recomende mantê-los como válvulas de escape quando for exigido um controle preciso sobre as dependências de efeitos. Testado massivamente na Meta e suportado pelo Expo SDK 54, Vite e Next.js, o compilador também remove uma categoria de boilerplate de desempenho que há muito tempo é um custo de uso em escala com

o React. O React 19 também introduz Actions e hooks como `useActionState` e `useOptimistic`, que simplificam a manipulação de formulários e as mutações de dados sem depender de bibliotecas externas. O lançamento da React Foundation sob a Linux Foundation em 2025 — com Amazon, Expo, Callstack, Microsoft, Software Mansion e Vercel se juntando à Meta — fortalece ainda mais a estabilidade de longo prazo da biblioteca e aborda uma preocupação que times cautelosos historicamente citavam ao considerar a adoção.

101. React Native

Adote

O React Native mudou para Adote como nossa escolha padrão para o desenvolvimento mobile multiplataforma. Embora estivesse anteriormente em Experimente, o lançamento da Nova Arquitetura — especificamente o `JSI` e o `Fabric` — resolveu preocupações de longa data em relação aos gargalos de ponte e à velocidade de inicialização. Nossos times observaram ganhos significativos de desempenho em transições complexas de UI e cargas de trabalho com uso intensivo de dados. Ao se afastar da ponte assíncrona, o React Native agora entrega uma responsividade comparável com as implementações nativas, mantendo uma única base de código. Nós o usamos com sucesso em múltiplos projetos em produção e descobrimos que o ecossistema em torno do Expo e do React é maduro e estável. Embora o gerenciamento de estado ainda exija um planejamento cuidadoso, os benefícios de produtividade dos workflows de `fast refresh` e os conjuntos de habilidades compartilhadas superam esses custos. Para a maioria dos casos de uso mobile híbridos, o React Native agora é nossa recomendação principal para times que buscam desempenho, consistência e velocidade.

102. Svelte

Adote

O Svelte é um framework de UI em JavaScript que compila componentes em código JavaScript otimizado no momento do *build*, em vez de depender de um grande *runtime* no navegador ou de um DOM virtual. Desde a última vez que o destacamos em Trial, vimos mais equipes utilizando-o com sucesso em produção. O SvelteKit também o tornou uma escolha mais robusta para SSR e aplicações web full-stack, aumentando nossa confiança em movê-lo para Adopt. Em nossa experiência, os motivos originais para escolher o Svelte continuam válidos: ele gera bundles pequenos, entrega alto desempenho em tempo de execução e oferece um modelo de componentes mais simples. Novos recursos do Svelte 5, como runes e snippets, tornam a reatividade e a composição de UI mais explícitas e flexíveis. Em comparação a frameworks de frontend mais pesados, o Svelte proporciona uma experiência de desenvolvimento mais limpa e com menos código. O feedback das equipes sugere cada vez mais que ele é uma alternativa sólida ao React ou Vue, e não apenas uma opção de nicho. Embora os times ainda devam considerar a familiaridade com o ecossistema, contratações e adequação à plataforma, recomendamos agora o Svelte como uma escolha padrão sensata para a construção de aplicações web modernas onde o desempenho e a simplicidade de entrega são prioridades.

103. Typer

Adote

O Typer é uma biblioteca Python para construir interfaces de linha de comando (CLIs) a partir de funções padrão anotadas com tipos, fornecendo texto de ajuda automático, autocompletar no shell e um caminho claro desde pequenos scripts até aplicações de CLI maiores. Estamos observando uma relevância crescente à medida que os times transformam ferramentas internas, automação

e workflows de pessoas desenvolvedoras adjacentes à IA em CLIs de primeira classe. Em nossa experiência, o Typer é fácil de introduzir em projetos reais, e os times apreciam a rapidez com que ele produz comandos claros e legíveis. Seus pontos fortes incluem APIs orientadas por dicas de tipo, ajuda e autocompletar automáticos e um caminho de baixo atrito desde scripts simples até CLIs de múltiplos comandos. No entanto, é uma solução específica para Python e pode não ser o melhor encaixe quando um comportamento de CLI altamente customizado ou consistência entre múltiplas linguagens for necessário. Recomendamos o Typer para times que constroem CLIs para entrega, operações e workflows de experiência da pessoa desenvolvedora.

104. Agent Development Kit (ADK)

Experimente

O Agent Development Kit (ADK) é o framework do Google para construir e operar agentes de IA com abstrações orientadas à engenharia de software para orquestração, ferramentas, avaliação e deployment. Seu ecossistema e capacidades operacionais amadureceram significativamente desde que o incluímos em Avalie, com desenvolvimento ativo em múltiplas linguagens e recursos mais fortes de observabilidade e runtime. Frameworks de agentes nativos de fornecedores (vendor-native) agora são um campo disputado, com o Microsoft Agent Framework, Amazon Bedrock AgentCore, o OpenAI Agents SDK e o Claude Agent SDK avançando como opções concorrentes. Alternativas de código aberto, como o LangGraph e o CrewAI, continuam sendo escolhas fortes, especialmente onde os times priorizam a portabilidade do framework e ecossistemas mais amplos. O ADK ainda está em fase pré-GA (anterior à disponibilidade geral) em algumas partes, com eventuais arestas (rough edges) e atritos de atualização, mas temos visto mais projetos usando-o com sucesso, particularmente aqueles de times que já investiram na plataforma do Google.

105. DeepEval

Experimente

O DeepEval é um framework baseado em Python e de código aberto para avaliar o desempenho de LLMs. Ele pode ser usado para avaliar sistemas de geração aumentada por recuperação (RAG) e aplicações construídas com frameworks como LlamaIndex ou LangChain, bem como para fazer o baseline e o benchmarking de modelos. O DeepEval vai além de métricas simples de correspondência de palavras, avaliando precisão, relevância e consistência para fornecer uma avaliação mais confiável em cenários do mundo real. Ele inclui capacidades como detecção de alucinação, pontuação de relevância de resposta e otimização de hiperparâmetros. Um recurso que nossos times acharam particularmente útil é que ele permite definir métricas personalizadas e específicas para cada caso de uso. Recentemente, o DeepEval se expandiu para suportar workflows complexos de agentes e sistemas conversacionais de múltiplos turnos. Além de avaliar os resultados finais, ele fornece métricas integradas para corretude da ferramenta, eficiência da etapa e conclusão da tarefa, incluindo a avaliação de interações com servidores MCP. Ele também introduz a simulação de conversas para gerar casos de teste automaticamente e fazer testes de estresse em aplicações de múltiplos turnos em escala.

106. Docling

Experimente

O Docling é uma biblioteca de código aberto em Python e TypeScript para converter documentos não estruturados em saídas limpas e legíveis por máquina. Usando uma abordagem baseada em visão computacional para a compreensão semântica e de layout, ele processa entradas complexas

— incluindo PDFs e documentos digitalizados — em formatos estruturados como JSON e Markdown. Isso o torna uma ótima opção para pipelines de geração aumentada por recuperação (RAG) e para produzir saídas estruturadas de LLMs, em contraste com abordagens de busca visual, como o ColPali. O Docling fornece uma alternativa de código aberto e auto-hospedável a serviços proprietários gerenciados em nuvem, como Azure Document Intelligence, Amazon Textract e Google Document AI, ao mesmo tempo que se integra bem a frameworks como o LangGraph. Em nossa experiência, ele tem um bom desempenho em cargas de trabalho de extração em escala de produção tanto em PDFs digitais quanto escaneados, incluindo arquivos muito grandes contendo texto, tabelas e imagens. Ele entrega um ótimo equilíbrio entre qualidade e custo para workflows downstream de RAG com agentes. Com base nesses resultados, estamos movendo o Docling para a seção Experimente.

107. LangExtract

Experimente

O LangExtract é uma biblioteca Python que usa LLMs para extrair informações estruturadas de textos não estruturados com base em instruções definidas pela pessoa usuária, com um embasamento preciso na fonte que vincula cada entidade extraída à sua localização no documento original. Ele processa materiais específicos de domínio, como notas clínicas e relatórios. Um ponto forte é a rastreabilidade da fonte, que garante que cada ponto de dado extraído possa ser rastreado de volta à sua origem. As entidades extraídas podem ser exportadas como um arquivo JSONL, um formato padrão para dados de modelos de linguagem, e visualizadas por meio de uma interface HTML interativa para revisão contextual. Times considerando saídas estruturadas de LLMs para processamento de documentos devem avaliar o LangExtract ao lado de abordagens de imposição de esquema, como o PydanticAI. O LangExtract é mais adequado para materiais de origem não estruturados de formato longo, enquanto o Pydantic AI se destaca em restringir formatos de saída para entradas mais curtas e previsíveis.

108. LangGraph

Experimente

Desde o último Radar, observamos que a arquitetura do LangGraph — que trata todo sistema de multiagentes como grafos stateful com um estado compartilhado global — nem sempre é a melhor abordagem para construir sistemas baseados em agentes. Também vimos uma abordagem alternativa, usada em frameworks como o Pydantic AI, que também funciona bem. Em vez de começar com um grafo rígido e um estado compartilhado massivo, essa abordagem favorece agentes simples se comunicando por meio da execução de código, com estruturas de grafo adicionadas posteriormente quando necessário. Isso frequentemente resulta em sistemas mais enxutos e mais eficazes para muitos casos de uso. Como cada agente só tem acesso ao estado de que precisa, o raciocínio, os testes e o debugging se tornam mais fáceis. Como resultado, tiramos o LangGraph de Adopt. Embora continue sendo uma ferramenta poderosa, não o vemos mais como a escolha padrão para construir todo sistema baseado em agentes.

109. LiteLLM

Experimente

O LiteLLM começou como uma fina camada de abstração sobre múltiplos provedores de LLM, mas desde então se expandiu para um gateway de IA completo. Além de simplificar a integração de API, ele aborda preocupações transversais comuns em sistemas de GenAI, como novas tentativas e failover, balanceamento de carga entre provedores e rastreamento de custos com controles de

orçamento. Nossos times estão adotando cada vez mais o LiteLLM como um padrão sensato para aplicações impulsionadas por IA. O gateway fornece um local consistente para implementar práticas de governança, incluindo rastreamento de solicitações, controle de acesso, gerenciamento de chaves de API e guardrails em nível de borda (edge-level), como filtragem de conteúdo e ocultação ou mascaramento de dados. No entanto, os times que dependem de recursos diferenciados de provedores descobrirão que estes frequentemente exigem parâmetros específicos do provedor, reintroduzindo o acoplamento que o gateway deve eliminar. Também vale notar que o modo `drop_params` descarta silenciosamente os parâmetros não suportados, o que significa que as capacidades podem ser perdidas nas decisões de roteamento sem visibilidade. O LiteLLM continua sendo uma escolha pragmática para o controle operacional, mas os times devem entender que apoiar-se em capacidades específicas de provedores significa manter tanto uma dependência de gateway quanto um código acoplado ao provedor.

110. Modern.js

Experimente

O Modern.js é um meta-framework React da ByteDance que estamos colocando em Avaliação (Trial) para times com requisitos de micro-frontend construídos no Module Federation. O gatilho é prático: o `nextjs-mf` está chegando ao fim da vida útil. O Pages Router receberá apenas pequenas correções retroativas, nenhum novo desenvolvimento está planejado e espera-se que os testes de CI sejam removidos até meados ou final de 2026. Com o Next.js carecendo de suporte oficial ao Module Federation e o plugin da comunidade sendo descontinuado, o time central do Module Federation agora recomenda o Modern.js como o principal framework suportado para arquiteturas baseadas em federação. O plugin `@module-federation/modern-js-v3` fornece a conexão automática de build pronta para uso, com streaming SSR e Bridge APIs disponíveis como capacidades separadas. No entanto, combiná-los tem limitações: o `@module-federation/bridge-react` ainda não é compatível com ambientes Node, tornando o Bridge inviável em cenários de SSR. Nossa experiência inicial é positiva, e o caminho de migração é bem definido para times que já usam o Module Federation. O ecossistema fora da ByteDance ainda está amadurecendo, então os times devem se planejar para uma documentação mais escassa e um engajamento mais próximo com o upstream. Esta continua sendo uma recomendação em Avaliação (Trial), porque o investimento se justifica para casos de uso de Module Federation onde não existe atualmente uma alternativa com melhor suporte.

111. Agent Lightning

Avalie

O Agent Lightning é um framework de otimização e treinamento de agentes que permite a otimização automática de prompts, fine-tuning supervisionado e aprendizado por reforço baseado em agentes. A maioria dos frameworks de agentes foca em construir agentes, não em melhorá-los ao longo do tempo. O Agent Lightning aborda isso permitindo que os times melhorem continuamente os agentes existentes sem alterar sua implementação subjacente, já que ele suporta frameworks como o AutoGen e o CrewAI. Ele alcança isso por meio de uma abordagem chamada Training-Agent Disaggregation (Desagregação de Agentes de Treinamento), que introduz uma camada entre os frameworks de treinamento e de agente. Essa camada consiste em dois componentes principais: o Lightning Server e o Lightning Client. O Lightning Server gerencia o processo de treinamento e expõe uma API para modelos atualizados, enquanto o Lightning Client atua como um runtime que coleta traces e os envia de volta ao servidor para apoiar o treinamento. Recomendamos que times com implantações de agentes já estabelecidas explorem o Agent Lightning como uma maneira de melhorar continuamente o desempenho dos agentes.

112. GitHub Spec Kit

Avalie

O desenvolvimento orientado a especificações (Spec-driven development) teve destaque em nossas discussões neste ciclo. Duas grandes abordagens estão surgindo: equipes que confiam na evolução contínua dos agentes de programação com o mínimo de estrutura, e aquelas que preferem fluxos de trabalho definidos e especificações detalhadas. Vários de nossos times estão experimentando práticas orientadas por especificações usando o GitHub Spec Kit, principalmente em ambientes pré-existentes (brownfield). Um conceito-chave no Spec Kit é a constituição, um documento-base de princípios que alinha o ciclo de vida de desenvolvimento de software. Na prática, os times relataram que uma constituição útil normalmente captura o escopo do projeto, contexto de domínio, versões de tecnologia, padrões de codificação e estrutura do repositório (por exemplo, arquitetura hexagonal ou módulos em camadas). Esse contexto compartilhado ajuda os agentes a operar dentro dos limites arquitetônicos pretendidos. Os times também encontraram desafios como o crescimento excessivo das instruções, onde a adição contínua de contexto do projeto levou a um conjunto crescente de instruções do agente e, eventualmente, à deterioração do contexto. Um time resolveu isso extraindo orientações reutilizáveis para arquivos de skills, mantendo as instruções do agente enxutas e carregando o contexto detalhado apenas quando necessário. Em sistemas brownfield, muito do retrabalho decorre de intenção pouco clara, suposições ocultas e descoberta tardia de restrições. Um time adotou um ciclo de vida de spec → plan → tasks → coding → review, o que ajudou a identificar problemas mais cedo. Com o tempo, também moveram o contexto reutilizável para arquivos como `.github/prompts/speckit.<command>.prompt.md`, tornando os prompts mais curtos e o comportamento do agente mais consistente. Os times relataram algumas limitações, incluindo verificações defensivas desnecessárias e saídas em Markdown verbosas demais que aumentavam a carga cognitiva. Personalizar os templates e as instruções do Spec Kit — por exemplo, limitando o número de arquivos markdown gerados e reduzindo a verbosidade do console — ajudou a resolver alguns desses problemas. Em última análise, pessoas engenheiras experientes, particularmente aquelas com boas práticas de clean code e arquitetura, tendem a extrair o máximo de valor de workflows orientados a especificações.

113. Mastra

Avalie

O Mastra é um framework nativo em TypeScript e de código aberto para construir aplicações e agentes de IA. Ele oferece um mecanismo de workflow baseado em grafos, acesso unificado a uma variedade de provedores de LLMs, suspensão e retomada com intervenção humana, bem como primitivas de RAG e de memória. Ele também inclui a criação de servidores MCP e ferramentas integradas para avaliação e observabilidade, com suporte de uma documentação clara para desenvolvedores. O Mastra fornece uma alternativa às stacks fortemente baseadas em Python, permitindo que os times construam capacidades de IA robustas diretamente em ecossistemas web já existentes, como Node.js ou Next.js. Vale a pena avaliá-lo para times que já apostam no ecossistema TypeScript e que desejam evitar ter de migrar para Python na camada de IA.

114. Pipecat

Avalie

O Pipecat é um framework de código aberto para a construção de agentes multimodais de voz em tempo real, com um modelo de pipeline modular para STT, LLM, TTS e orquestração de transporte. Ele está atraindo grande interesse porque os times podem iterar rapidamente no comportamento conversacional e trocar de provedores com um atrito relativamente baixo. Em comparação com o

LiveKit Agents, o Pipecat oferece maior flexibilidade de framework, mas um caminho de produção menos integrado, especialmente para implantação auto-hospedada, confiabilidade de transporte e tratamento de turnos (turn handling) de baixa latência em escala. Nós colocamos o Pipecat em Avalie porque ele fornece uma base sólida voltada para a engenharia, mas é necessário um trabalho significativo de engenharia de plataforma antes que se possa confiar nele para cargas de trabalho de produção críticas para o negócio.

115. Superpowers

Avalie

Com o uso crescente de agentes de programação, não existe um único fluxo do trabalho prescrito para todos os times. Em vez disso, os times estão desenvolvendo fluxos de trabalho sob medida com base em seu contexto e restrições. Nossos times em todo o mundo estão ativamente definindo e experimentando esses fluxos de trabalho de programação assistida por IA. O Superpowers é um desses fluxos construído sobre skills combináveis. Ele envolve um agente de programação em um fluxo de trabalho estruturado de skills que incentiva o brainstorming antes da programação, planejamento detalhado antes da implementação, desenvolvimento orientado a testes (TDD) com ciclos red-green-refactor obrigatórios, debugging sistemático focado primeiro na causa raiz e revisão de código pós-implementação. O Superpowers é distribuído como um plugin por meio do Claude Code plugin marketplace, bem como do plugin marketplace do Cursor.

116. TanStack Start

Avalie

O TanStack Start é um framework full-stack construído sobre o TanStack Router para React e Solid. Ele é comparável ao Next.js, suportando SSR, caching e muitos dos mesmos recursos. O TanStack Start fornece segurança em tempo de compilação de ponta a ponta em funções de servidor, carregadores e roteamento, reduzindo o risco de links quebrados ou formatos de dados incompatíveis no frontend. O framework favorece a configuração explícita em vez da convenção, tornando a experiência mais próxima de trabalhar com o React puro. Você também pode adicionar capacidades de SSR progressivamente, conforme necessário. Em comparação com o Next.js, que tem padrões mais diretivos que podem levar a comportamentos inesperados se os times não estiverem familiarizados com seu funcionamento interno, o TanStack Start é mais explícito e previsível. O ecossistema TanStack também amadureceu significativamente, oferecendo um forte conjunto de ferramentas para construir aplicações web modernas.

117. TOON (Token-Oriented Object Notation)

Avalie

O TOON (Token-Oriented Object Notation) é uma codificação legível por pessoas para dados JSON projetada para reduzir o uso de tokens quando dados estruturados são passados para LLMs. Ele permite que os times mantenham o JSON em sistemas existentes e o convertam apenas no momento da interação com o modelo. Isso é importante porque o custo de tokens, a latência e as restrições da janela de contexto estão se tornando fatores concretos de projeto em pipelines de RAG, workflows de agentes e outras aplicações intensivas em IA. O JSON puro costuma consumir tokens com chaves repetidas e overhead estrutural em vez de conteúdo útil. Em nossa avaliação inicial, o TOON é uma otimização interessante de “última milha” para entradas de prompt, particularmente para grandes conjuntos de dados regulares onde um formato mais orientado ao schema (schema-aware) pode ser tanto mais eficiente quanto mais fácil para os modelos processarem do que o JSON. Ele não é

um substituto para JSON em APIs, bancos de dados ou saídas do modelo, e frequentemente é a escolha errada para estruturas profundamente aninhadas ou não uniformes, arrays semi-uniformes ou dados tabulares simples, sem hierarquia onde o CSV é mais compacto. Ele também pode ser menos adequado para fluxos sensíveis à latência onde o JSON compacto tem um bom desempenho. Por esses motivos, acreditamos que vale a pena avaliar o TOON para times que constroem aplicações de LLM em que o tamanho da entrada estruturada tem impacto relevante em custo ou qualidade. Os times devem compará-lo com JSON ou CSV por meio de benchmarks usando seus próprios dados e stack de modelos.

118. Unsloth

Avalie

O Unsloth é um framework de código aberto para *fine-tuning* (ajuste fino) de LLMs e aprendizado por reforço, focado em tornar o treinamento significativamente mais rápido e eficiente em termos de memória. O ajuste fino de LLMs envolve bilhões de multiplicações de matrizes, cargas de trabalho que se beneficiam da aceleração por GPU. O Unsloth otimiza essas operações traduzindo-as em *kernels* personalizados altamente eficientes para GPUs NVIDIA, reduzindo drasticamente o custo e o uso de memória. Isso permite realizar o *fine-tuning* de modelos em GPUs de consumo, como a T4 e superiores, em vez de exigir clusters H100 caros. O Unsloth suporta LoRA, ajuste fino completo, treinamento multi-GPU e *fine-tuning* de contexto longo (até 500 mil tokens) para modelos populares como Llama, Mistral, DeepSeek-R1, Qwen e Gemma. À medida que as aplicações de IA para domínios específicos dependem cada vez mais do ajuste fino, o Unsloth reduz significativamente a barreira de entrada.

Mantenha-se atualizada com os artigos e informações relacionados ao Radar

Assine o Technology Radar para receber e-mails mensais com insights de tecnologia da Thoughtworks e divulgações dos futuros volumes.

Assine