技术雷达

有态度的前沿技术解析

/thoughtworks

第 28 期

关于技术雷达

Thoughtworker 酷爱技术。我们为每个人建造技术，研究技术，测试技术，开源技术，书写技术，并不断致力于改进技术。我们的使命是支持卓越软件并掀起 IT 革命。我们创建并分享 Thoughtworks 技术雷达就是为了支持这一使命。由 Thoughtworks 中一群资深技术领导组成的 Thoughtworks 技术顾问委员会创建了该雷达。他们定期开会讨论 Thoughtworks 的全球技术战略以及对行业有重大影响的技术趋势。

技术雷达以独特的形式记录技术顾问委员会的讨论结果，从首席技术官到开发人员，雷达为各路利益相关方提供价值。这些内容只是简要的总结。

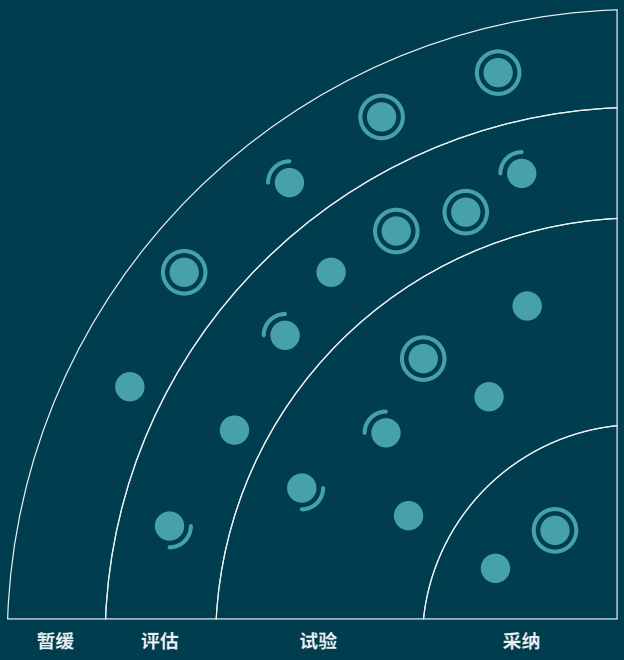
我们建议您探究这些技术以了解更多细节。技术雷达的本质是图形性质，把各种技术项目归类为技术、工具、平台和语言和框架。如果技术可以被归类到多个象限，我们选择看起来最合适的一个。我们还进一步将这些技术分为四个环以反映我们目前对其的态度。

想要了解更多技术雷达相关信息，请点击：
http://thoughtworks.com/cn/radar/faq

雷达一览

技术雷达持续追踪有趣的技术是如何发展的，我们将其称之为条目。在技术雷达中，我们使用象限和环对其进行分类，不同象限代表不同种类的技术，而环则代表我们对它作出的成熟度评估。

软件领域瞬息万变，我们追踪的技术条目也如此，因此您会发现它们在雷达中的位置也会改变。



采纳：我们强烈主张业界采用这些技术。我们会在适当时候将其用于我们的项目。

试验：值得追求。重要的是理解如何建立这种能力。企业应该在风险可控的项目中尝试此技术。

评估：为了确认它将如何影响你所在的企业，值得作一番探究。

暂缓：谨慎推行

新的　挪进 / 挪出　没有变化

技术雷达是具有前瞻性的。为了给新的技术条目腾出空间，我们挪出了近期没有发生太多变化的技术条目，但略去某项技术并不表示我们不再关心它。

贡献者

技术顾问委员会（TAB）由 Thoughtworks 的 21 名高级技术专家组成。TAB 每年召开两次面对面会议，每两周召开一次视频会议。技术顾问委员会由 Thoughtworks 首席技术官 Rebecca Parsons 组建。

作为一个综合型组织，TAB 能够审视影响 Thoughtworks 技术和技术人员的各种主题。本期技术雷达内容基于 2023 年 3 月的 TAB 线上会议创建。


Rebecca Parsons（CTO）


Martin Fowler（Chief Scientist）


Bharani Subramaniam


Birgitta Böckeler


Brandon Byars


Camilla Falconi Crispim


Erik Dörnenburg


Fausto de la Torre


Hao Xu


Ian Cartwright


James Lewis


Marisa Hoenig


Maya Ormaza


Mike Mason

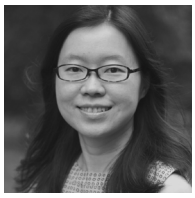

Neal Ford


Pawan Shah


Scott Shaw


Selvakumar Natesan


Shangqi Liu


Sofia Tania


Vanya Seth

中国区技术雷达汉化组：

陈锋 | 李辉 | 娄麒麟 | 严嘉阳 | 边晓琳 | 程博 | 程显通 | 董翔锐 | 樊卓文 | 冯炜 | 符雨菡 | 高晓明 | 高玉山 | 郭晨 | 何蜜 | 何蔚 何向东 | 蒋亦雄 | 李光正 | 李康宁 | 李思远 | 李天舒 | 李妍 | 梁晶 | 廖燊 | 卢冠昀 | 马宇航 | 任旭东 | 沈哲宇 | 苏晓风 | 孙郁俨 索云清 | 陶慧 | 田鹏 | 童圣 | 王启瑞 | 王芹芹 | 王浠婵 | 肖森元 | 邢砚敏 | 邢砚敏 | 熊欣 | 杨阳 | 于海源 | 余琦 | 张丽 | 张双海 张旭海 | 郑茗蔓

本期主题

实用人工智能的飞速崛起

别多想，这篇主题文章并非由 ChatGPT 撰写。人工智能已经在专业领域默默酝酿了几十年，像 GitHub Copilot 这样的工具在几年前就已经存在（并逐渐被采用）了。然而，在过去几个月里，类似 ChatGPT 这样的工具已经彻底改变了人们对人工智能的认识，并使得这类工具开始被广泛使用。本期技术雷达中有几个条目涉及到了 AI 在项目上的实际应用，不仅仅是关于代码建议的范畴：基于 AI 辅助的测试先行开发、使用 AI 帮助构建分析模型等。类似于电子表格能够让会计师停止用计算器手动重新计算复杂表格，下一代的 AI 会承担起替换需要知识（而不是智慧）的乏味任务，来解放技术从业者包括开发人员。

不过，我们提醒不要过度或不当使用人工智能。现今，AI 模型能够生成一个良好的初稿。但生成的内容始终需要由人类监测、验证、审查和负责任地使用。如果忽视这些预警，机构和用户可能面临名誉和安全风险。甚至有些产品用例也提醒用户，“AI 生成的内容可能存在错误。在使用前请确保它的正确性和合理性”。

易用的无障碍设计

多年来，无障碍设计一直是组织重视的因素。最近，我们着重展示了我们的团队在工具和技术方面经验的增长，这些经验使开发具备了更好的无障碍设计。我们几个地区的团队通过宣传活动增强了对这些技术的认知。我们在持续集成流水线开发、设计中的无障碍注解、智能引导无障碍测试、静态检查和单元测试方面，提出了与无障碍设计相关的条目。我们很愿意看到人们越来越重视这个主题，为更多的人提供改进后的功能访问方式，这些技术毫无疑问是优秀的。

Lambda 陷阱

无服务器函数 AWS Lambdas 越来越频繁地出现在架构师和开发者的工具箱中，并被用于实现各种基于于云基础架构的任务。然而，就像许多有用的东西一样，有时候解决方案开始时简洁实用，但随着不断成功、持续演进，最终违反范式中规定的约束、变得沉重不堪，终遭弃用。在看到许多无服务器风格解决方案成功应用的同时，我们也从项目中听到了许多警示性的故事，比如 Lambda 弹球反模式。虽然出现了很多解决这些问题的工具，但是这些工具极易被误用。例如，帮助 Lambda 之间共享代码或协调复杂互动的工具，它可能可以解决一个简单常见的问题，但随着新的构建模块的加入就会面临出现低劣架构模式的风险。如果您需要一个工具来管理跨无服务器函数集合的代码共享和独立部署，那么也许是时候重新思考该方法的适用性了。像所有的技术解决方案一样，无服务器有其适宜的应用场景，但它的许多功能在使用时都需要权衡利弊，这种情况随着解决方案的演进会更加突出。

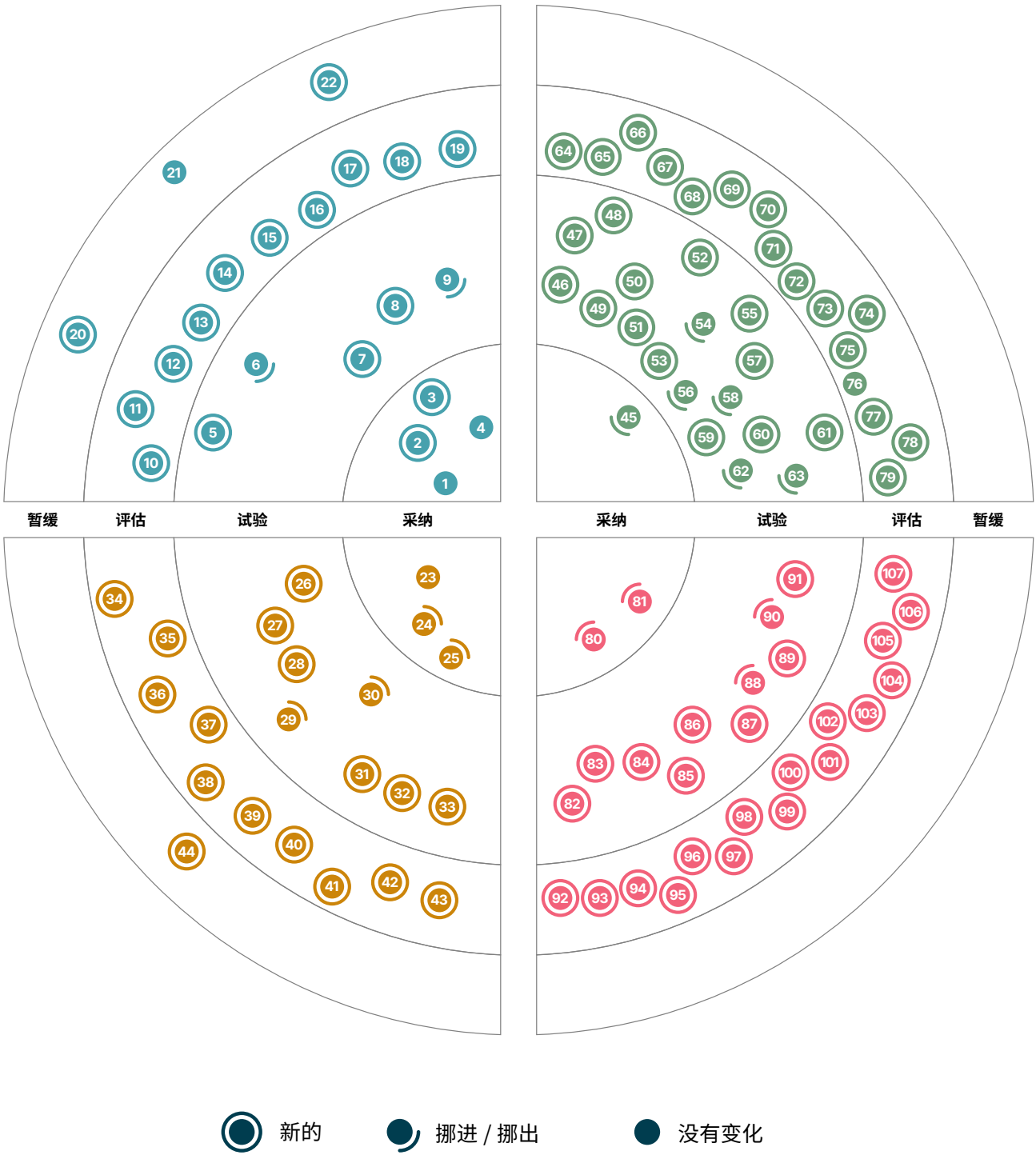
数据分析和人工智能中的工程严谨性

我们始终认为“质量内建”是开发可靠的数据分析与机器学习模型的重要因素。测试驱动的数据转化，数据健全测试和数据模型测试使数据流水线可以更有力的支持分析系统。机器学习系统需要模型验证和质量控制来确保结果符合伦理，平等无偏见。通过整合这些实践，企业能更好地利用人工智能和机器学习，基于各自的用户群体打造负责任、数据驱动的解决方案。相关的工具生态也在不断演进，日趋成熟。例如，Soda Core 是一个在接收数据伊始就开始验证并自动监测异常的数据质量工具，Deepchecks 结合了持续集成和模型检查，将好的工程实践融入分析系统。Giskard 对 AI 模型进行质量控制，可以检查模型的偏见等其他负面因素，这与我们提倡的在开发 AI 解决方案时关注伦理的理念相符。我们认为这些逐渐成熟的工具可以证明，数据分析、人工智能与良好的工程实践相结合正在成为主流。

声明，还是编程?

每次技术雷达讨论时都会发生的，好像永无止境的话题在这次会议变得格外激烈——对于给定的任务，应该使用 JSON、YAML 或者像 HCL 这样的领域特定语言来编写声明性规范，还是应该在通用编程中编写代码语言？例如，我们讨论了 Terraform Cloud Operator 与 Crossplane 的区别，在其他情况下对部署流水线进行编程时，是否使用 AWS CDK 或 Dagger。声明性规范虽然通常更易于阅读和编写，但其有限的抽象会导致重复代码的产生。适当的编程语言可以使用抽象来避免重复，但这些抽象会让代码变得更难理解，特别是当抽象经过多年的变化而分层时。根据我们的经验，上述问题没有标准答案。这两种方法团队都应该考虑，当一个解决方案无法用一种语言类型去简洁的实现时，就应该重新评估使用另一种类型。有些时候拆分关注点并用不同的语言实现它们会更加合理。

The Radar



The Radar

技术

采纳

1. 将产品管理思维应用于内部平台
2. 将 CI/CD 基础设施作为一种服务
3. 精简软件依赖项
4. 将运行成本作为架构健康度的考量

试验

5. 设计中的无障碍注解
6. 有限界的使用低代码平台
7. 对仅提供 API 的产品的演示前端
8. Lakehouse 架构
9. 可验证凭证

评估

10. 无障碍意识的组件测试设计
11. 基于 AI 辅助的测试驱动开发
12. 领域特定的大型语言模型
13. 智能引导无障碍测试
14. 使用 Logseq 构建团队知识库
15. 提示工程
16. 测试基础设施时的可达性分析
17. 自托管式大型语言模型
18. 管理技术健康状况优先于技术债务
19. CI/CD 的零信任保护

暂缓

20. 对 Webhooks 的管理不够严谨
21. Lambda 弹球
22. 竭尽全力的计划

平台

采纳

23. Contentful
24. GitHub Actions
25. K3s

试验

26. Apache Hudi
27. 云上 Arm
28. Ax
29. DuckDB
30. 特征库
31. RudderStack
32. Strapi
33. TypeDB

评估

34. Autoware
35. Cozo
36. Dapr
37. Immuta
38. Matter
39. Modal
40. Neon
41. OpenLineage
42. Passkeys
43. Spin

暂缓

44. Denodo 作为主要的数据转换工具

The Radar

工具

采纳

45. DVC

试验

46. Akeyless
47. Apicurio Registry
48. EventCatalog
49. FOSSA
50. Gitleaks
51. Helmfile
52. IBM Equal Access Accessibility Checker
53. Ktlint
54. Kubeflow
55. Mend SCA
56. Mozilla SOPS
57. Ruff
58. Soda Core
59. Steampipe
60. Terraform Cloud Operator
61. TruffleHog
62. Typesense
63. Vite

评估

64. axe Linter
65. ChatGPT
66. DataFusion
67. Deepchecks
68. 设计令牌翻译工具
69. Devbox
70. Evidently
71. Giskard
72. GitHub Copilot
73. iamlive
74. Kepler
75. Kubernetes External Secrets Operator
76. Kubeshark
77. Obsidian
78. Ory Kratos
79. Philips 的自我托管 GitHub 运行器

暂缓

—

语言和框架

采纳

80. Gradle Kotlin DSL
81. PyTorch

试验

82. dbt 单元测试
83. Jetpack CameraViewfinder
84. Jetpack DataStore
85. Mikro ORM
86. 按应用设定的语言偏好设置
87. Quarto
88. River
89. Stencil
90. Synthetic Data Vault
91. Vitest

评估

92. .NET 7 Native AOT
93. .NET MAUI
94. dbt-expectations
95. Directus
96. Ferrocene
97. Flutter 嵌入式平台
98. Fugue
99. Galacean Engine
100. LangChain
101. mljar-supervised
102. nanoGPT
103. pandera
104. Qwik
105. SolidJS
106. Turborepo
107. WebXR 设备 API

Hold 暂缓

—

技术

采纳

1. 将产品管理思维应用于内部平台
2. 将 CI/CD 基础设施作为一种服务
3. 精简软件依赖项
4. 将运行成本作为架构健康度的考量

试验

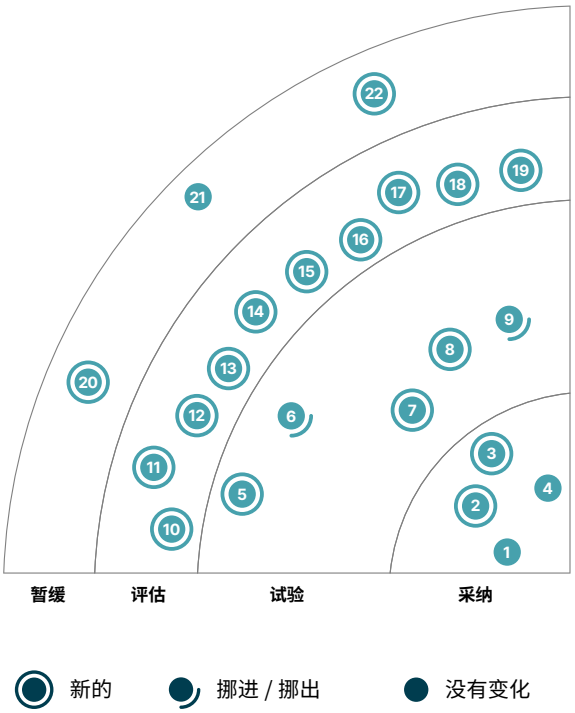
5. 设计中的无障碍注解
6. 有限界的使用低代码平台
7. 对仅提供 API 的产品的演示前端
8. Lakehouse 架构
9. 可验证凭证

评估

10. 无障碍意识的组件测试设计
11. 基于 AI 辅助的测试驱动开发
12. 领域特定的大型语言模型
13. 智能引导无障碍测试
14. 使用 Logseq 构建团队知识库
15. 提示工程
16. 测试基础设施时的可达性分析
17. 自托管式大型语言模型
18. 管理技术健康状况优先于技术债务
19. CI/CD 的零信任保护

暂缓

20. 对 Webhooks 的管理不够严谨
21. Lambda 弹球
22. 竭尽全力的计划



1. 将产品管理思维应用于内部平台

采纳

我们持续从那些将产品管理思维应用于内部平台的团队获得良好的反馈。不过，要记住一个关键特征：这不只是关于团队结构或重命名已有的平台团队；它还涉及到在团队中应用以产品为中心的工作实践。具体来说，我们收到的反馈表明，除非团队具有以产品为中心的思维方式，否则他们在使用此技术时将面临挑战。这可能意味着需要额外的角色，比如产品经理，以及对其他领域的改变，比如需求收集和对成功的衡量。以这种方式工作意味着与内部消费者（开发团队）建立同理心并且在设计上与他们合作。平台产品经理制定路线图并确保平台为业务带来价值和增强开发人员的体验。我们会继续将这项技术视为构建内部平台的关键，以求快速而高效地推出新数字解决方案。

2. 将 CI/CD 基础设施作为一种服务

采纳

将 CI/CD 基础设施作为一种服务已经是很多元化以及成熟的方案，以至于需要自己管理整个 CI 基础设施的情况变得非常少见。使用 GitHub Actions、Azure DevOps 或 Gitlab CI/CD 等管理服务，具有托管云服务的所有常见优势（和权衡）。您不需要花时间、精力和硬件成本来维护和运营这个通常很复杂的基础设施。对于自己托管 CI 设施的公司，虽然团队可以受益于弹性和自助服务，然而往往在配置更多合适的代理或获得新的插件或功能时遇到瓶颈。即使是需要在自己的硬件上运行构建和验证的用例，现在也大多可以用自我托管的运行器来满足需求（我们已经写过一些关于 GitHub Actions 的文章，actions-runner-controller 和 Philips 的自我托管 GitHub 运行器）。然而，请注意，使用托管服务并不意味着您可以轻易获得安全保障；虽然成熟的服务提供了所需要的所有安全功能，但仍然需要实施对 CI/CD 基础架构的零信任安全。

3. 精简软件依赖项

采纳

初始工具包和模板被广泛用于软件项目以加快初始设置，但它们可能会为项目引入许多不必要的依赖项。实践依赖裁剪很重要——定期仔细检查这些依赖并剔除未使用的依赖，这有助于减少构建和部署时间，并且可以通过移除潜在漏洞来降低项目受攻击的风险。尽管这不是一项新技术，但鉴于对软件上下游的攻击频率越来越高，我们提倡重新关注它。

4. 将运行成本作为架构健康度的考量

采纳

自动估算、跟踪和预测云基础设施运行成本对于今天的组织至关重要。云服务商精心设计的定价模型，加上价格参数的不断增多以及持续变化的架构，可能会导致运行成本超出预算。尽管这种技术已经在 2019 年后被采用，我们仍然想强调考虑将运行成本作为架构健康度的考量的重要性，特别是在云服务被广泛采用和 FinOps 实践越来越受到关注的今天。许多商业平台提供了工具来协助企业的负责人人整合并说明云服务的成本。其中一些旨在向财务机构或原始业务单位显示云服务运行成本。然而，云服务消费决策通常是在系统设计时做出。因此

在做出设计决策时使用一些方法来预测其架构决策对成本产生的影响就显得很重要。一些团队会在开发进程的初期通过自动化方式预算成本。像 Infracost 这样的工具可以帮助团队在考虑更改“基础设施即代码”时预测成本影响，这种计算可以自动化执行，并纳入持续集成（CD）流程中。请注意，成本将受架构决策和实际使用水平的影响；要做到这一点，您需要对期望使用水平进行审慎地预测。及早和频繁地反馈运行成本可以防止其失控。当预测成本偏离了预期或可接受的范围时，团队可以讨论是否是时候进一步调整架构了。

5. 设计中的无障碍注解

试验

在软件交付中越早考虑无障碍，就能越简单、更低成本的保证交付物对尽可能多的人可用。设计中的无障碍注释工具能促进沟通，帮助团队从工作的开始就考虑到文档结构、语义化的 HTML 和替代文本等重要的元素。这使得团队确保用户界面符合国际无障碍标准，并解决那些实际上很容易避免的常见无障碍问题。Figma 提供了一系列的无障碍性注释插件：The A11y Annotation Kit，Twitter 的 Accessibility Annotation Library 和 Axe 的工具集 Axe for Designers。

6. 有限界的使用低代码平台

试验

我们一直倡导写尽可能少的代码。简洁是我们在软件开发时笃信的核心价值之一。例如，我们尽量不预测未来的需求，只实现满足当前业务需求的代码，而不考虑其他内容。创建工程平台是一个在组织层面上实现这个目标的可能方法。

这也是现在许多流行的低代码平台的既定目标。像 Mendix 或 Microsoft Power Apps 这些平台可以展示通用的业务流程以方便重复使用，并简化了部署新功能和交付给用户的问题。近年来，这些平台在可测试性和对良好工程实践的支持方面取得了巨大的进步。它们特别适用于简单的任务或事件触发的应用程序。然而，要求它们适应几乎无限的业务需求会带来复杂性。虽然开发人员可能会少写（或不写）代码，但他们也必须成为一个全面的商业平台的专家。我们建议企业考虑他们是否需要这些产品带来的所有功能，或者他们是否更应该追求限界的低代码平台，无论是开发自己的作为内部产品的平台，还是通过谨慎限制商业低代码平台的使用范围，使其仅限完成于它们擅长的简单任务。

7. 对仅提供 API 的产品的演示前端

试验

捕捉并传达 API 的业务价值是开发 API 时的一大挑战。就其本质而言，API 是一种技术产品。尽管开发人员可以轻松理解 JSON 数据、OpenAPI（Swagger）规范和 Postman 演示，但业务利益相关者会更倾向于可交互的 API 产品演示。通常当我们能够实际看到并亲身体验产品时，产品的价值会更加清晰地表达出来。也正是因为这样，我们有时会认为投资对仅提供 API 的产品的演示前端是值得的。当定制化界面 UI 与 API 产品一起构建时，业务利益相关者可以将产品与他们可能更熟悉的纸质表格或报告进行类比。随着交互模型和 UI 演示的丰富性不断发展，业务利益相关者可以对 API 产品的发展方向做出更明智的决策。基于 UI 工作也有额外的益处，它

可以增加开发人员对业务用户的同理心。虽然 API 产品的演示前端并不是一项新技术——当 API 产品出现时，我们就能在必要时构建演示前端。然而，由于这项技术并不广为人知，我们仍然认为该技术值得关注。

8. Lakehouse 架构

试验

Lakehouse（数据湖仓一体）架构是一种数据湖的可扩展性与数据仓库可靠性和性能相结合的架构风格，它使得组织能在一个平台上使用类似于 SQL 的工具和技术，储存分析大量不同的数据，而不是将他们放在单独的在数据湖和数据仓库层中。尽管这个术语（LakeHouse）经常与作为供应商的 Databricks 之类的厂商有关，像 Delta Lake，Apache Iceberg 和 Apache Hudi 之类的开源方案也值得考虑。Lakehouse 架构可以补充数据网格的实现。自治的数据产品团队可以在他们的数据产品选择使用 Lakehouse。

9. 可验证凭证

试验

三年前，当我们第一次在雷达中提及可验证凭证（VC）时，它是一个引人注目的标准，有着一些潜在的应用前景，但在爱好者群体之外并没有广为人知。对于如州政府等负责实施该标准的证书授予机构，情况更是如此。三年疫情之后，随着加密安全、尊重隐私和机器可验证电子证书的需求增长，政府开始意识到可验证凭证的潜力。W3C 标准以凭证持有者为中心，这与我们使用物理凭证时的体验相似：用户可以将他们可验证的凭证放到自己的数字钱包中，并在任何时候向任何人展示，而无需得到凭证发行者的许可。这种去中心化的方式有助于用户更好地管理和有选择地披露自己的信息，大大提高了数据隐私的保护。在过去的 6 个月里，我们的一些团队参与了使用可验证凭证技术的项目。毫不意外，不同国家和政府部门的情况并不相同。我们的团队在多个项目上探索了去中心化标识、可验证凭证和可验证展示的不同组合。这是一个仍在发展的领域，现在我们有了更多的经验，我们希望在雷达中持续跟踪它。

10. 无障碍意识的组件测试设计

评估

在软件交付过程中，需要提早考虑无障碍设计的地方有很多，Web 组件测试是其中环节之一。像 chai-a11y-axe 这样的测试框架插件在其 API 中提供了断言，以检查基本的无障碍设计。但是，除了使用测试框架所提供的功能外，无障碍意识组件测试设计进一步提供了屏幕阅读器和其他辅助技术所需的所有语义元素。

首先，不要使用 test id 或 class 来寻找和选择你要验证的元素，而是使用通过 ARIA 角色或其他辅助技术使用的语义属性来识别元素。一些测试库，如 Testing Library，甚至在文档中直接推荐这样做。其次，不要只测试点击交互，还要考虑不能使用鼠标或看不到屏幕的用户，并考虑增加对键盘和其他交互的测试。

11. 基于 AI 辅助的测试驱动开发

评估

和软件行业的许多人一样，我们一直在探索快速演进的 AI 工具，以帮助我们编写代码。我们看到很多人通过将实现代码输入 ChatGPT，然后请求其生成测试代码。但作为 TDD（测试驱动开发）的坚定支持者，我们并不

想经常将可能包含敏感信息的实现代码输入到外部模型中，因此我们尝试了基于 AI 辅助的测试驱动开发的技术。在这种方法中，我们让 ChatGPT 为我们生成测试代码，然后由开发人员来实现功能。具体而言，我们首先在一个可在多个用例中重复使用的提示“片段”中描述我们使用的技术栈和设计模式。然后，我们描述我们想要实现的具体功能，包括验收标准。基于这些信息，我们要求 ChatGPT 生成在我们的架构风格和技术栈中实现这一功能的实现计划。一旦我们对这个实现计划进行了合理性检查，我们就要求它为我们的验收标准生成测试代码。

这种方法对我们来说效果出奇的好：它要求团队提供对其架构风格的简明描述，帮助初级开发人员和新团队成员编写符合团队现有风格的功能特性。这种方法的主要缺点是，尽管我们没有向模型提供源代码，但我们仍然向其输入了可能包含敏感信息的技术栈和功能描述。至少在这些 AI 工具的“商业版”面世之前，团队应确保与法律顾问合作，以避免任何知识产权问题。

12. 领域特定的大型语言模型

评估

在之前的技术雷达中，我们已经提到过 BERT 和 ERNIE 之类的大型语言模型（LLMs）；但是领域特定的大型语言模型是一个新兴的趋势。使用领域特定数据对通用大型语言模型进行微调能将它们用于各种各样的任务，包括信息查询，增强用户支持和内容创作。这种实践已经在法律和金融领域展现出现它的潜力，例如使用 OpenNYAI 进行法律文书分析。随着越来越多的组织开始对大型语言模型进行试验和越来越多像 GPT-4 这样的新模型的发布，我们预期大型语言模型在不久的将来会有更多领域特定的用例。但是使用大型语言模型仍面临许多挑战和缺陷。首先，大型语言模型能“很自信地犯错”，所以需要构建一些机制来保证结果的准确性。其次，第三方大型语言模型可能保留或二次分享你的数据，这会对保密信息和数据的所有权带来风险。组织应当仔细审阅使用条款和供应商的可信度，或考虑在自己控制的基础设施上训练和运行大型语言模型。就像其他的新技术一样，在业务上使用大语言模型前需要保持谨慎，并理解采用大型语言模型带来的后果和风险。

13. 智能引导无障碍测试

评估

对于从未使用过辅助技术，并且觉得对像 Web 内容无障碍指南（WCAG）这样的标准仍然一无所知的人来说，使网页应用程序兼容辅助技术可能会有些困难。智能辅助无障碍测试是这样一类可以帮助测试已正确实现无障碍设计的工具，而无需成为无障碍技术专家。这些工具是浏览器扩展，它们会扫描网站，总结辅助技术会如何解释网页，然后通过一组问题以确认您创建的结构和元素是否符合预期。我们已经在一些项目中使用过 axe DevTools、Web Accessibility Insights for Web 或 ARC Toolkit 等工具。

14. 使用 Logseq 构建团队知识库

评估

团队知识管理是一个熟悉的概念，团队使用诸如维基（wikis）等工具来存储信息和培训新团队成员。我们的一些团队现在更倾向于将 Logseq 作为团队知识库。Logseq 是一种开源的知识管理系统，由图形数据库驱动，帮助用户组织想法、笔记和创意，并可以通过基于 Git 的存储方式适应团队使用。Logseq 允许团队构建一个公开透明的知识库，为每个成员提供个性化的学习路径，促进高效的启动培训。然而，与任何知识管理工具一样，团队需要对他们的知识库进行良好的归纳整理，以避免信息过载或组织混乱。

虽然类似的功能也可以在像 Obsidian 这样的工具中找到，但 Logseq 的关键区别在于其注重于知识的使用，基于段落的链接使团队成员能够快速找到相关的上下文，而无需阅读整篇文章。

15. 提示工程

评估

提示工程（Prompt engineering）指的是为生成式 AI 模型设计和优化提示的过程，以获得高质量的模型响应。这个过程包括精心设计特定、清晰易懂和与所需任务或应用相关的提示，以引导模型输出有用的结果。提示工程旨在增强大型语言模型（LLM）在问题回答、算术推理任务或特定领域上下文中的能力。在软件开发中，我们可以使用提示工程让 LLM 根据与利益相关者的简要对话或一些笔记撰写故事、API 或测试套件。培养有效的提示技巧正在成为处理人工智能系统的重要技能。提示工程被一些人认为是一门艺术，而另一些人则认为它是一门科学。同时我们也需要考虑到其潜在的安全风险，例如“提示注入攻击”。

16. 测试基础设施时的可达性分析

评估

在部署基础设施代码时，我们发现很多时间会花费在诊断和修复因系统之间无法相互通信导致的生产问题上。由于它们之间的网络拓扑结构可能非常复杂，即使已经正确配置了单个端口和端点，整个路由也可能无法完全遍历。基础设施测试实践通常包括验证正确的端口是开放还是关闭，或者某个端点是否可以被访问，但我们最近才开始在测试基础设施时进行可达性分析。该分析通常涉及比简单的是 / 否判断更多的内容。例如，工具可能会遍历并报告通过中转网关的多个路由。此技术得到了所有主要云供应商工具的支持。Azure 有一个名为网络观察程序的服务，可以在自动化测试中进行脚本编写，而 GCP 则支持连通性测试。而现在，在 AWS 中，您可以测试同一组织内跨账户的可达性。

17. 自托管式大型语言模型

评估

大型语言模型通常会运行在具有强大的 GPU 的基础设施上。我们如今可以看到一些大型语言模型的移植版本，比如 llama.cpp，这些模型能在不同的硬件上运行，包括 Raspberry Pi（树莓派）、笔记本电脑和通用服务器等。因此自托管式大型语言模型已经成为现实。目前，有许多开源的自托管式大型语言模型，如 GPT-J、GPT-JT 和 LLaMA。自托管这种方式有许多好处，比如可以更好地控制模型在一些特定使用场景的微调、提高安全性和隐私性，以及支持离线访问。不过在决定使用自托管这种方式之前，您应该仔细评估组织的能力和运行此类大型语言模型需要消耗的成本。

18. 管理技术健康状况优先于技术债务

评估

在软件交付组织中，如何管理技术债务是一个经久不衰的话题。哪些是技术债务而哪些不是？如何确定它的优先级？最重要的是，如何向内部利益相关者展示偿还技术债务的价值？遵循敏捷宣言的推理方式——“尽管右项

有其价值，我们更重视左项的价值”，我们喜欢“管理技术健康状况优先于技术债务”的想法。澳大利亚的 REA 团队分享了一个很好的案例，介绍了他们如何追踪管理开发、运维和架构三个方面的系统健康状况。

将关注点放在技术健康状况而非技术债务上更有建设性。这样将团队与减少技术债务的最终价值联系起来，并帮助团队确定优先级。理想情况下，每个被解决的技术债务都应与某项已达成共识的期望相关联。团队应将技术健康状况评级与其他服务级别目标（SLO）一样对待，并在某个类别的健康评级跌出“安全区域”时优先进行改进。

19. CI/CD 的零信任保护

评估

如果没有得到正确的安全配置，运行构建和交付的流水线的基础设施和工具可能成为一个大麻烦。流水线需要访问关键数据和系统比如源代码，凭证和密码来构建和部署软件。这让这些系统非常容易吸引恶意攻击者。因此，我们强烈推荐引入零信任安全机制来保障 CI/CD 流水线和基础设施——尽可能少地信赖它们。这包含一系列技术：如果可行，根据你的云供应商通过联合身份校验机制如 OIDC 来验证你的流水线，而不是直接给他们获取机密数据的权限。实现最小特权原则，最小化个人用户和执行账户的权限，而不是创建一个拥有无限访问权限的全能账户。用临时的方式去使用任务执行器而不是复用他们，来减少暴露先前任务的秘密或在受到攻击的运行器上运行任务的风险。保证软件在你的代理服务器和任务执行器上是最新的。像监控你的生产环境软件一样监控你的 CI/CD 系统的完整性，保密性和可用性。我们不断见到有团队会忘记这些实践，特别是当他们使用在内网自我管理的 CI/CD 基础设施的时候。所有这些实践在不仅在内网中都很重要，在使用托管服务时，因为扩大了攻击面和影响范围，这种实践会变得更加重要。

20. 对 Webhooks 的管理不够严谨

暂缓

随着远程工作的增加，聊天协作平台和 ChatOps 的采用也在增加。这些平台通常提供 Webhook（网络挂钩）作为自动发送消息和通知的简单方式，但我们关注到一个令人担忧的趋势：对 Webhooks 的管理不够严谨—将它们视为配置而不是秘密或凭据。这可能会导致钓鱼攻击和内部信息泄露。

Webhook 是提供对内部空间的特权访问的凭据，可能包含可以轻松提取和直接使用的 API 密钥。不将它们视为密钥会导致钓鱼攻击的发生。在 Git 仓库中的 Webhook 可以轻松提取并用于发送有效的欺诈信息，用户可能没有任何身份验证的方式。为了缓解这种威胁，处理 Webhook 的团队需要改变他们的习惯，并将 Webhook 视为敏感凭据。软件开发人员与 ChatOps 平台构建集成必须注意到这种风险，确保这些 Webhook 具备适当的安全措施。

21. Lambda 弹球

暂缓

虽然无服务器架构对解决一些问题非常有用，但它们确实有一定的复杂性，尤其是当它们涉及到复杂执行和跨多个相互依赖的 Lambdas 的数据流时——这有时会导致所谓的 Lambda 弹球架构。我们团队发现维护和测试 Lambda 弹球架构可能非常有挑战：理解基础设施、部署、诊断和调试都会变得困难。在代码层面上，根本不可能将领域概念和所涉及的多个 Lambdas 之间做简单映射，这使得任何改变和添加都具有挑战。尽管我们相信无服务器是适合某些问题和领域的，但它并不是每个问题的“万能解法”，这就是为什么你应该尽量避免陷入 Lambda 弹球。一种有助于解决这个问题的方法，就是区分公共接口（public interface）和已发布接口（published interface），并在它们之间应用带有已发布接口的领域边界。

22. 竭尽全力的计划

暂缓

虽然在产品管理社区中，应该预留一些交付能力余量是众所周知的，但我们发现仍然有许多团队会为团队制订竭尽全力的计划。在迭代计划中预留一定的产能，往往可以提高可预测性以及更好的质量。因为这不仅有助于增强团队应对意外事件（如疾病、生产问题、意外的产品请求和技术债务）的弹性，还让团队可以进行提升生产力的活动，比如团队建设和创意构思，从而推动产品创新。留有一定的交付余量，可以让团队更加专注于所生产的软件的健壮性，并更加关注正确的度量信息。我们的经验表明，就像高速公路的车辆过多就会导致缓慢而令人沮丧的交通一样，充分利用团队的能力往往会导致生产效率的崩溃。当我们的一个团队需要处理不可预测的支持请求时，他们只按照开发能力的 2/3 来计划功能交付的速度，从而增加了 25% 的交付量，并减少了 50% 的周期时间波动。

平台

采纳

23. Contentful
24. GitHub Actions
25. K3s

试验

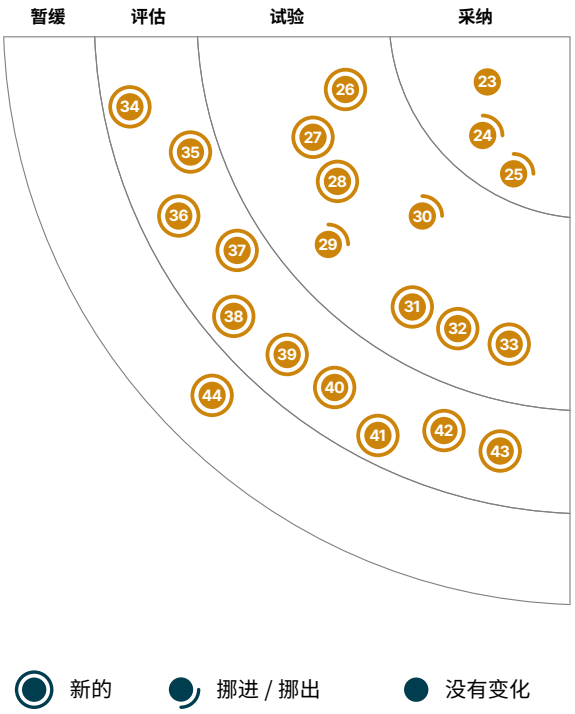
26. Apache Hudi
27. 云上 Arm
28. Ax
29. DuckDB
30. 特征库
31. RudderStack
32. Strapi
33. TypeDB

评估

34. Autoware
35. Cozo
36. Dapr
37. Immuta
38. Matter
39. Modal
40. Neon
41. OpenLineage
42. Passkeys
43. Spin

暂缓

44. Denodo 作为主要的数据转换工具



23. Contentful

采纳

无头内容管理系统已经成为数字平台的常见组成部分，Contentful 仍然是我们在这个领域的首选，但像 Strapi 这样的新进入者也给我们留下了深刻的印象。我们尤为喜欢 Contentful 的 API 优先的工作方式以及实现了 CMS as code。它支持强大的代码化内容模型原语和内容模型演化脚本，这使得它可以像其他数据存储模式一样被处理，并演进式数据库设计实践可以应用于 CMS 开发。最近，Contentful 发布了应用程序框架来编写应用，以便更容易地将 Contentful 适应个人业务流程并与其他服务集成。这些应用程序可以由特定组织建立，也可以为特定组织构建，与此同时市场上的应用也在快速增多。

24. GitHub Actions

采纳

对于许多需要在新环境中快速启动和运行 CI 或 CD 的团队来说，GitHub Actions 已经成为了默认选择。此外，它还能承载更复杂的工作流程，并且能够调用其他复合任务中的任务。尽管 GitHub 应用市场中的生态在不断发展，我们仍然强烈建议谨慎授予第三方 GitHub Actions 权限访问您的构建流水线。我们建议您遵循 Github 有关安全强化的建议，避免以不安全的方式共享机密信息。但是，直接在托管源代码的 GitHub 上创建构建工作流的便利性，再加上能使用像 act 等开源工具在本地运行 GitHub Actions，让其成为简化团队设置和新成员上手流程的一个引人注目的选项。

25. K3s

采纳

K3s 仍是我们在边缘计算和资源受限环境下的默认 Kubernetes 发行版。它轻量，完全兼容 Kubernetes，且操作开销较少。它使用 sqlite3 作为默认存储引擎，而不是 etcd。由于它在一个进程中运行所有相关组件，因此具有减少内存占用的优点。我们已将 K3s 用于工控系统和 POS 机环境中，我们对自己的决定非常满意。K3s 的运行时 containerd 已经支持 wasm，因此 K3s 可以直接运行和管理 WebAssembly 负载，进一步减少了运行时开销。

26. Apache Hudi

试验

Apache Hudi 是一款开源数据湖平台，它将 ACID 事务保障引入了数据湖中。我们尝试了在大数据量高吞吐量场景下使用 Hudi 进行实时写入和更新数据，取得了非常理想的效果。特别令人满意的是，Hudi 对定制压缩算法的支持很灵活，这有助于解决“小文件”读写的问题。Apache Hudi 和 Delta Lake 以及 Apache Iceberg 都是同类的工具，它们提供相似的功能，但是在底层的实现方式和功能细节上又各有千秋。

27. 云上 Arm

试验

云上 Arm 计算实例在过去几年中越来越受欢迎。因为与传统的基于 x86 的实例相比，其成本更低，能源效率也更高。如今 AWS、Azure 和 GCP 等许多云供应商都提供基于 Arm 的实例。云上 Arm 的成本优势对于运行大型工作负载或需要扩容的企业来说特别有利。根据我们的经验，除非依赖于特定架构，建议所有工作负载都使用 Arm 计算实例。多架构 docker 镜像等很多支持多架构的工具也可以简化构建和部署的工作流。

28. Ax

试验

在面对探索大型配置空间的挑战时，可能需要大量的时间来评估给定配置，团队可以转向自适应实验，这是一种机器引导的迭代过程，以资源高效的方式找到最佳解决方案。Ax 是一个用来管理和自动化自适应实验的平台，它包括机器学习实验，A/B 测试和模拟。目前，它支持两种优化策略，一种是使用 BoTorch 的贝叶斯优化，它是建立在 PyTorch 基础上的，以及 contextual bandits。Facebook 在发布 Ax 和 BoTorch 时描述了例如提高后端基础设施的效率，调整排名模型和优化机器学习平台的超参数搜索的用例。我们已经在多种用例中有了丰富的 Ax 应用经验，虽然超参数调整工具是存在的，我们尚未发现有能提供与 Ax 在一定范围上具有相似功能的平台。

29. DuckDB

试验

DuckDB 是一个用于数据科学与数据分析的嵌入式列式数据库。数据分析师通常会在本地将数据加载进诸如 pandas 或 data.table 这些工具中，从而可以在于服务器内扩展解决方案前就做到快速分析模式和形成假设。然而，我们现在使用 DuckDB 来处理这些用例，因为它释放出了比内存分析更大的潜力。DuckDB 支持庞大事务的 range joins，向量化执行和多版本并发控制（MVCC），我们的团队对此表示非常满意。

30. 特征库

试验

任何软件都需要正确表示其所应用的那一领域，并且应该始终了解关键的目标。机器学习项目也不例外。特征工程是机器学习软件系统工程和设计的重要部分。特征库是一个架构上的概念，能促进识别、发现和监测与给定领域或业务问题有关的特征。实现这一概念需要结合架构设计，数据工程和基础设施管理，来创建一个可扩展的、高效的、可靠的机器学习系统。从工具的角度看，您可以找到开源的和完全托管的方案，但这些只包含了这个概念的一部分。在端到端的机器学习系统设计中，实现特征库带来了以下能力：（1）定义准确特征的能力；（2）增强数据的可复用性并且让特征在不同模型中保持一致和可用的能力，其中还包括设置特征工程管道，以规划特征库中的数据的能力；（3）帮助特征发现的能力和（4）提供特征服务的能力。我们的团队利用特征库获得了这些对端到端机器学习系统的便利。

31. RudderStack

试验

RudderStack 是一个可以轻松存储数据至数据仓库或数据湖的客户数据平台（CDP）。这种方法越来越多地被称为无头 CDP（Headless CDP），它将 CDP 的功能和用户界面分离，强调可通过 API 灵活配置，同时强调数据仓库 / 数据湖为主要存储方式。正如人们对此类产品所期望的那样，RudderStack 既拥有丰富的类库支持与第三方服务集成（可同时作为源和接收器），也能够接受自定义事件。RudderStack 既有商业产品，也有部分功能受限的自托管 OSS 版本。

32. Strapi

试验

Strapi 是一个基于 Node.js 的开源无头内容管理系统（CMS），它类似于 Contentful。它已经出现了一段时间，我们也成功将其用于部分项目。Strapi 提供了 REST 和 GraphQL 两种类型的 API，具有丰富的文档，易用的数据模型 API，并且支持界面和逻辑的定制化。

33. TypeDB

试验

TypeDB 是一个知识图谱数据库，设计用于处理复杂的数据关系，从而简化查询和分析大型数据集。TypeDB 的 TypeQL 查询语言类似于 SQL 语法，它可以简化模式定义、查询和探索的学习曲线。TypeDB 带有各种工具，包括一个命令行界面和一个图形用户界面，这使其更容易与数据库集成，TypeDB Studio 提供了一些集成 TypeDB 的功能，如管理模式，查询数据，可视化关系，甚至与他人合作。它有大量的文档，并且有一个活跃的支持社区。我们的团队用它来建立跨不同数据库的分类学概念知识图谱，并通过增加新的推理规则来利用其强大的推理能力来提高效率、减少工作量。凭借其友好的开发者体验和有帮助的社区，TypeDB 是一个很好的候选者，任何团队，如果在实施依赖复杂数据关系的解决方案，都可以考虑这个工具，包括自然语言数据、推荐引擎和知识图谱。

34. Autoware

评估

Autoware 是一个基于 ROS（机器人操作系统）开发的，开源的自动驾驶软件栈。它可以用于为各种各样的车辆，如汽车和卡车等，开发和部署先进辅助驾驶系统（ADAS）。它不仅为自动驾驶的各个组成方面提供了一套开发工具与算法，譬如车身感知，驾驶决策和控制；而且它也内置有路线规划和控制的模块，可以基于周围环境和目的地的信息为车辆提供路线信息。它的出现促进了自动驾驶技术的开放式创新。我们在 Autoware 上构建自动驾驶原型去验证新产品构想，并发现它卓有成效。

35. Cozo

评估

Cozo 是一个以 Datalog 作为查询语言的可嵌入关系型数据库。其对时间旅行（time-travel）查询及对在关系型数据模式中建模图数据的支持使我们非常感兴趣。我们很喜欢它将数据存储委托给现有的主流引擎，例如 SQLite，RocksDB，Sled 和 TiKV。尽管 Cozo 还处在早期开发阶段，我们仍认为它值得一试。

36. Dapr

评估

Dapr 是分布式应用程序运行时（Distributed Application Runtime）的缩写，帮助开发人员构建在云中运行的具有弹性、无状态和有状态微服务。一些人可能会将其与服务网格（service mesh）混淆，因为它使用一个边车（Sidecar）架构，作为应用程序旁边独立运行的进程。Dapr 更加面向应用程序，并专注于封装构建分布式应用所需的容错性和连接性。例如，Dapr 提供多个构建块（building blocks），从服务调用和消息发布 / 订阅到分布式锁定等都是分布式通信中常见的模式。我们团队最近在一个项目上评估了 Dapr；基于这一成功经验，他们期待将其引入未来其他项目中。

37. Immuta

评估

Immuta 是一个数据安全平台，它允许你安全地访问数据，自动发现敏感数据并审计数据在组织中的使用情况。在过去，当我们考虑安全风险时，我们已经谈到了自动化、工程实践和将安全策略代码化的重要性。数据安全也不例外。我们的团队一直在探索 Immuta，它能将数据访问策略作为代码来管理，以实现精细化的访问控制，这超出了基于角色的访问控制（RBAC）所能提供的范围。基于版本控制的策略可以被测试，然后配置为 CI/CD 管道的一部分。在一个去中心化的数据生态系统中，比如由数据网格促成的生态系统，拥有特定领域的角色会导致身份认证系统中的角色或用户组扩散。Immuta 的基于属性的访问控制（ABAC）的特性将访问授权简化为一个数学方程式，即把用户的“属性”与数据源的“标签”相匹配。这个平台虽然很新，但在满足数据安全需求方面绝对值得考虑。

38. Matter

评估

Matter 是由亚马逊、苹果、谷歌、康卡斯特和 Zigbee 联盟（现已更名为 Connectivity Standards Alliance，简称 CSA）联合推出的智能家居技术的开放标准。它使设备能够与任何已通过 Matter 认证的生态系统配合使用，从而减少了不同供应商的设备和物联网平台之间的碎片化，促进了互操作性。它专注于在应用层面上的标准化、支持 Wi-Fi 和 Thread 作为通信媒介。与 Zigbee 等其他协议不同，Matter 得到了主要科技公司的支持。尽管支持 Matter 的设备数量仍然相对较少，但它在物联网领域日益重要，因此对于那些希望构建智能家居和物联网解决方案的人来说，它是值得评估的。

39. Modal

评估

Modal 是一种平台即服务（PaaS），您无需自己的基础设施即可按需计算。Modal 让您可以部署机器学习模型、大规模并行计算作业、任务队列和 Web 应用程序。它提供了容器抽象，使得从本地部署到云端部署无缝切换，在本地和云端都可以热重载。它甚至可以自动删除部署，避免了手动清理的需要，但也可以使其持久化。

Modal 由为 Spotify 开发第一个推荐引擎的同一个团队编写。它负责端到端的人工智能（AI）/ML 栈，并能提供按需的 GPU 资源，如果您有特别密集的计算需求，这将非常有用。无论您是在笔记本电脑上还是在云端工作，Modal 都能发挥作用，为运行和部署您的项目提供了一种简单高效的方式。

40. Neon

评估

Neon 是 AWS Aurora PostgreSQL 的一款开源替代产品。云原生分析型数据库已经采用了将存储与计算节点分离的技术，以便根据需要弹性扩展。然而，在事务性数据库中进行相同的操作比较困难。Neon 通过其新的多租户存储引擎技术实现了这一点。它对主流 PostgreSQL 代码进行最小的更改，并利用 AWS S3 进行长期数据存储，在计算方面弹性扩缩容（包括缩放到零）。这种架构有几个好处，包括便宜和快速地克隆、写入时复制和分支。我们对基于 PostgreSQL 的创新非常兴奋。我们的团队正在评估 Neon，并建议您也对其进行评估。

41. OpenLineage

评估

OpenLineage 是一个开放的数据管道沿袭元数据收集标准，旨在在作业运行时对其进行编整。它使用一致的命名约定定义了运行、作业和数据集实体的通用模型。沿袭模型的核心是可扩展的，可以通过定制切面来增加实体。OpenLineage 解决了生产者和消费者之间互通问题，否则大家不得不想各种办法实现互通。虽然存在它会成为另一个“中间标准”的风险，但作为 Linux 基金会 AI 和数据基金会项目，它获得被广泛采用的机会很大。OpenLineage 目前支持多个平台的数据采集，如 Spark、Airflow 和 dbt，但用户需要自己配置监听器。OpenLineage 对数据消费者的支持目前较为有限。

42. Passkeys

评估

“密码的终结”可能终于要到来了。在 FIDO 联盟的指导和苹果、谷歌、微软的支持下，passkeys 的可用性正在接近其他主流身份验证方式。当用 passkeys 注册新的登录信息时，它会产生一对密钥：网站收到公钥而用户保留私钥。它使用非对称加密处理登录。用户需要证明他们拥有私钥，但与密码不同，私钥不会被发送到网站上。在用户的设备上，会使用生物识别技术或 PIN 码来保护对 passkeys 的访问。

Passkeys 可以在各大软件生态中存储和同步，例如苹果的 iCloud 钥匙链、谷歌的密码管理器或微软的 Windows Hello。在大多数情况下，该功能只适用于最新的操作系统和浏览器版本。值得注意的是，Windows 10 并不支持在 Windows Hello 中存储密码。不过幸运的是，客户端到认证器协议（CTAP）让 passkeys 可以被保存在创建密钥或需要密钥登录的设备之外的不同设备上。例如，一个用户在 Windows 10 上为一个网站创建了一个 passkeys，并通过扫描二维码将其存储在 iPhone 上。因为密钥是通过 iCloud 同步的，所以用户可以从他们的 MacBook 等其他 Apple 生态设备上登录到该网站。Passkeys 也可以存储在硬件安全密钥上，在 iOS 和 Android 上也提供了对原生应用的支持。

尽管仍有一些可用性问题——例如，登录时需要设备支持并开启蓝牙，因为扫描二维码时要检查设备是否接近——passkeys 依然是值得考虑的。我们建议你在 passkeys.io 上试用，以了解它们的可用性。

43. Spin

评估

Spin 是一个开源平台，用于在 WebAssembly（WASM）中构建和运行微服务。在技术雷达以前的版本中，我们曾谈论过在浏览器中使用 WebAssembly，但现在由于它在细粒度的沙盒功能、跨语言互操作性和热重载的能力，我们正在目睹其在服务器端的渗透。通过使用 Spin CLI，您可以创建和分发基于 Rust、TypeScript、Python 和 TinyGo 的 WebAssembly 微服务。我们对 Spin 充满期待，并建议您仔细评估它，因为它正在退出早期预览阶段。

44. Denodo 作为主要的数据转换工具

暂缓

Denodo 是一款数据虚拟化工具，可以从单个平台将来自多个底层数据源和各种接口、经过转换后对使用者友好的数据方便地进行展示和保护。在 Denodo 中，可以使用类似于 SQL 的语言 VQL 创建虚拟数据库和视图，来实现数据转换。这些视图和虚拟数据库会在用户查询虚拟数据库时执行。在底层，Denodo 可以将虚拟数据库的查询委托给一个或多个底层数据库。

尽管 Denodo 可以简化展示消费者友好的数据的工作，但随着视图和虚拟数据库层的叠加，特别当查询需要连接多个底层数据库时，性能会受到影响。只有对产品行为和性能调优方案有相当深入的了解，才能解决这些问题。由于这些缺点，以及对单元测试支持有限，我们不建议使用 Denodo 作为主要数据转换工具，而应使用 Spark 或 SQL（通过 dbt）等工具做数据转换。

工具

采纳

45. DVC

试验

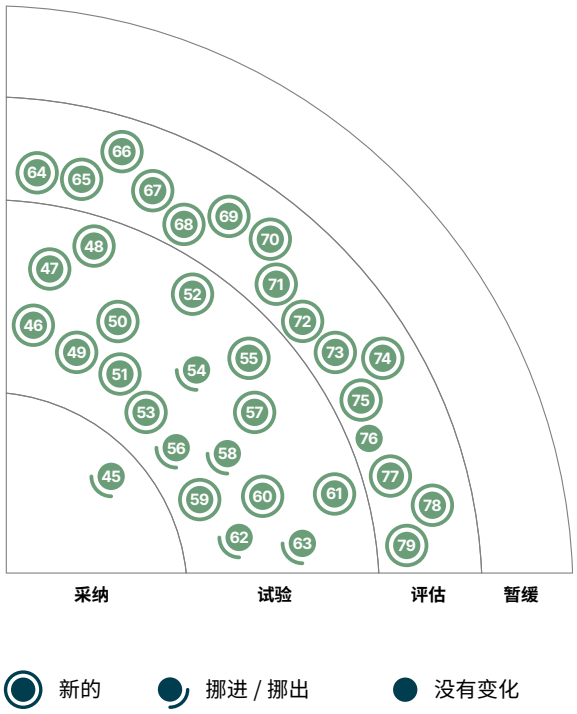
46. Akeyless
47. Apicurio Registry
48. EventCatalog
49. FOSSA
50. Gitleaks
51. Helmfile
52. IBM Equal Access Accessibility Checker
53. Ktlint
54. Kubeflow
55. Mend SCA
56. Mozilla SOPS
57. Ruff
58. Soda Core
59. Steampipe
60. Terraform Cloud Operator
61. TruffleHog
62. Typesense
63. Vite

评估

64. axe Linter
65. ChatGPT
66. DataFusion
67. Deepchecks
68. 设计令牌翻译工具
69. Devbox
70. Evidently
71. Giskard
72. GitHub Copilot
73. iamlive
74. Kepler
75. Kubernetes External Secrets Operator
76. Kubeshark
77. Obsidian
78. Ory Kratos
79. Philips 的自我托管 GitHub 运行器

暂缓

—



45. DVC

采纳

DVC 一直是我们在数据科学项目中管理实验的首选工具。由于 DVC 是基于 Git 的，因此对于软件开发人员来说，DVC 无疑是一个备感熟悉的环境，他们可以很容易地将以往的工程实践应用于数据科学生态中。DVC 使用其特有的模型检查点视图对训练数据集、测试数据集、模型的超参数和代码进行了精心的封装。通过把可再现性作为首要关注点，它允许团队在不同版本的模型之间进行“时间旅行”。我们的团队已经成功地将 DVC 用于生产环境，实现了机器学习的持续交付（CD4ML）。DVC 可以与任何类型的存储进行集成（包含但不限于 AWS S3、Google Cloud Storage、MinIO 和 Google Drive）。然而，随着数据集变得越来越大，基于文件系统的快照可能会变得特别昂贵。当底层数据发生快速变化时，DVC 借由其良好的版本化存储特性可以追踪一段时间内的模型漂移。我们的团队已经成功地将 DVC 应用于像 Delta Lake 这样的数据存储格式，利用它优化了写入时复制（COW）的版本控制。我们大多数的数据科学团队会把 DVC 加入到项目的“Day 0”任务列表中。因此，我们很高兴将 DVC 移至采纳。

46. Akeyless

试验

随着越来越多的组织采用云计算，许多组织开始同时集成多个云提供商，以最大限度地提高灵活性并最小化供应商锁定。然而，跨多个云提供商的密钥管理和访问控制可能会导致复杂性和安全风险增加，从而成为一项重大挑战。Akeyless 是一个基于云的集中化平台，提供统一的密钥管理，在管理密钥和敏感数据方面具有一系列优势。它能够与不同的云提供商无缝集成，简化了密钥管理和访问控制，以监测和控制谁可以访问敏感数据；通过加密、访问控制、多因素身份验证和其他安全机制，确保只有授权用户才能访问敏感数据。此外，它还提供了一个直观的界面用于管理和监控，这为开发和管理人员带来了更简单且更可扩展的体验。

47. Apicurio Registry

试验

在任何组织中，API 的生产者和使用者都需要就他们之间通信所使用的模式保持同步。特别是随着 API 数量以及相关生产者和使用者人数在组织中的增长，最初在团队间传递模式的简单做法将面临挑战。面对这个问题，我们的一些团队使用了 Apicurio Registry，这是一个开源的、集中式的注册表，可以存储各种类型的模式和 API 文档，包括 OpenAPI 规范，Protobuf 和 Avro 模式。Apicurio Registry 允许用户通过 UI，REST API 和 Maven 插件等方式进行交互。它还有规定模式演进限制的选项，比如说向后兼容性。此外，当使用 Kafka 客户端时，Apicurio Registry 能与 Confluent Schema Registry 兼容。虽然我们的团队发现 Confluent Schema Registry 的文档更有帮助，但 Apicurio Registry 满足了对各种模式的真实来源的需求。

48. EventCatalog

试验

现在企业通常使用事件流作为真实数据的来源，并在微服务架构中作为信息共享机制。这就需要标准化事件类型并在企业范围内共享这些标准。通常企业会部署事件模式注册表，但现有的解决方案往往只适用于单个代理，

比如 Apache Kafka 或 Azure Event Hub。此外，这些现有解决方案并不能提供超出简单模式定义的关于事件类型的详细文档。EventCatalog 是一个开源项目，为企业提供了一种广泛可访问的文档库，用于描述事件在业务种扮演的角色，它们在业务领域模型中的位置，以及哪些服务订阅和发布这些事件。如果您正在寻找一种向组织发布事件文档的方式，那么使用 EventCatalog 工具可能会避免您自己构建文档库的麻烦。

49. FOSSA

试验

FOSSA 是一个开源合规性工具，可以帮助开发人员和团队确定他们的代码依赖哪些开源组件，以及这些组件是根据哪些许可发布的。这些信息对于确保遵守各种开源许可证和维护软件物料清单十分重要。FOSSA 可以与各种技术栈的依赖管理工具集成，以识别项目中使用了哪些开源组件。还可以根据组织的政策突出显示任何许可证问题，并生成对应的报告。此外，FOSSA 的一些关键特性还包括与开发工作流（如 CI）集成，以及实时监控合规性。我们的许多团队和客户都认为 FOSSA 是一个有用且高效的工具。

50. Gitleaks

试验

Gitleaks 是一个开源 SAST（静态应用安全测试）命令行工具，用于检测 Git 仓库以防止把密码、API 密钥和访问令牌等机密信息硬编码到代码中。它可以用于 Git 的 pre-commit hook 和 CI/CD 流水线。我们团队发现，Gitleaks 比其他一些密码扫描工具更灵敏。Gitleaks 使用正则表达式和 entropy coding 字符串编码检测机密信息。在我们的经验中，entropy coding 提供了灵活的自定义正则表达式，允许团队基于他们的需求对机密信息进行更好的分类。例如，相较于把所有的 API 密钥笼统地归类为“通用型 API 密钥”，entropy coding 允许把密钥归类到“云服务提供商密钥”这种特定分类中。

51. Helmfile

试验

Helmfile 是一款开源命令行工具和声明式的标准，用于安装和管理多个 Helm chart，帮助您进行 Helm 配置文件、使用的 chart 等变更的版本管理。它使 Helm chart 能用于 CI/CD 工作流，有助于创建可重现环境。我们已经在使用 Helmfile 来管理那些包含大量 Helm chart 的复杂部署，并且发现它简化了我们的部署工作流。

52. IBM Equal Access Accessibility Checker

试验

缺陷发现得越早，修复成本更小。这就是为什么我们总是试图以静态分析、单元测试或在本地环境中运行的端到端测试等方式，尽可能快地向开发者提供反馈。无障碍设计也不例外，这就是我们过去介绍 Lighthouse、axe-core 和 axe Linter 等工具的原因。当涉及到已经部署在生产环境中的网页时，我们的一个团队选择了 IBM Equal Access Accessibility Checker 来进行直接的比较。虽然我们还在评估结果，但我们可以说，它提供了一种有效

的方法来测试已经发布了的网页。我们想要强调的是，这个工具应该用来增强而不是取代开发者的早期自动测试。该工具是在创作共用许可证（Creative Commons license）下发布的，在符合协议规定时可以免费使用。

53. Ktlint

试验

随着 Kotlin 生态系统的持续发展，我们的团队报告了使用 Ktlint 的良好体验：这是一个用于 Kotlin 代码，简单且易于配置的 linter 和 formatter。我们喜欢有态度的代码格式化工具（Opinionated and automated code formatting），因为这能让开发者更关注代码的行为，而不是它的外观；这个工具使开发团队能够高效的维护代码库的一致性和可读性，减少因格式问题而导致的混乱合并的可能性。Ktlint 可以很容易地配置在 pre-commit hook 中，它只针对有变化的文件，从而使集成过程更快。

54. Kubeflow

试验

Kubeflow 是一个 Kubernetes 原生的机器学习（ML）平台，它能简化模型生命周期中在不同基础设施上的构建、训练和部署流程。我们已经大量使用它的 Pipelines 来编码多个模型的包括实验、训练、服务用例的 ML 工作流。除 Pipelines 外，Kubeflow 还带有许多其他的组件，我们发现其中用于超参数调优的 Katib 组件以及多租户组件都是非常有用的。

55. Mend SCA

试验

Mend SCA（之前称为 Whitesource）可以用于检测开源软件依赖项，以识别它们是否最新、是否含有安全漏洞，或存在特殊许可证需求。我们的团队在将 Mend SCA 集成到生产流程方面有着不错的经验。无论是从 IDE 集成还是从 CI/CD 流水线集成中识别问题并自动提出 PR，Mend SCA 都提供了出色的开发体验。其他一些流行的 SCA 工具，如 Snyk，也值得探索，以满足你的安全需求。

56. Mozilla SOPS

试验

当谈到密钥管理的时候，我们总是建议密钥与代码解耦。然而，团队却时常面临着取舍权衡，一种方式是基于基础设施即代码思想的全自动化，另一种方式是使用一些手动步骤和诸如 vaults 的工具去管理、生成、更新密钥。例如，我们的团队使用 SOPS 工具生成构建基础设施所需要的根密钥。然而在某些情况下，从遗留代码仓库中移除密钥并不现实。对于这类需求，我们发现 Mozilla SOPS 是一个很好的工具，可以用来加密文本文件中的密钥。SOPS 集成了诸如 AWS、KMS、Azure Key Vault 等密钥仓库作为待加密密钥源。它支持跨平台运行，也支持 PGP 密钥。

57. Ruff

试验

Ruff 是一个新的 Python linter。我们认为，使用 linter 是毋庸置疑的，只需要考虑使用哪个 linter，因为 Python 提供了很多选择。Ruff 能够脱颖而出有两个原因：开箱即用的体验，以及性能。它内置了 500 多条规则，可以轻松取代 Flake8 和它的许多插件。我们的经验证实了 Ruff 团队对其性能的说法。它确实比其他 linter 快出至少一个数量级，这是一个巨大的优势，有助于减少大型代码库的构建时间。

58. Soda Core

试验

Soda Core 是一个开源数据质量与可观测性工具。我们的团队已经使用它来验证数据在到达系统之前和之后的转换，并设置自动化监测检查以检测异常情况。我们对 Soda Core 中用于编写数据检查的 DSL——SodaCL 非常满意 . SodaCL 能帮助除了数据工程师以外的其他团队成员来编写质量检查。总的来说，我们在解决大规模数据问题时使用 Soda Core 的体验非常好。

59. Steampipe

试验

Steampipe 是一个可以让你使用 SQL 实时查询 AWS，Azure，以及 GCP 等云服务的开源工具。Steampipe 拥有 100 多个插件以及内置的创建仪表板功能，使连接实时云配置数据和内外部数据集以及创建安全合规的仪表板变得非常简单。我们非常喜欢 Steampipe，并已经使用 AWS 云配置创建了几个此类仪表板。

60. Terraform Cloud Operator

试验

越来越多的团队在使用 Kubernetes Operators 模式来管理 Kubernetes 集群。为此，我们曾推荐过 Crossplane，现在有了另外一种选择——Terraform Cloud Operator。该工具通过扩展 Kubernetes 控制平面来集成 Terraform Cloud 和 Kubernetes，以通过 Kubernetes manifest 对云和本地基础设施进行生命周期管理。我们的团队使用它来提供资源，不管是 Kubernetes namespace，RoleBindings，云数据库实例，还是其他 SaaS 资源。我们非常喜欢它，因为它利用我们更熟悉的抽象层 Terraform 模块来操作云资源。

61. TruffleHog

试验

TruffleHog 是一款开源的静态应用程序安全测试工具（static application security testing，SAST），用于在各类源码中检测密钥信息。除了对 GitHub 和 GitLab 等最为常见的代码仓库进行扫描，TruffleHog 还能扫描 S3 和 GCS 等云存储桶、本地文件、本地目录以及 CircleCI 日志。开发人员可以将 TruffleHog 配置为 pre-commit hook 脚本，也可以直接使用它扫描一个 GitHub 组织的全部代码仓库历史记录。TruffleHog 支持自定义的正则语句模版，即使在当前的 alpha 阶段，该功能也十分易用。虽然 TruffleHog 有企业版本，但是我们发现开源版

本的 TruffleHog 更易于配置和使用，并且覆盖了绝大多数的常见场景。TruffleHog 拥有一个非常活跃的社区，时不时就会有新的功能更新。

62. Typesense

试验

Typesense 是一款可容错的开源搜索引擎，它针对高性能和低延迟的搜索体验进行了优化。如果您正在构建对延迟有较高要求的搜索应用，而且索引的大小可以载入内存，Typesense 将是个很好的选择。我们团队在高可用的多节点集群里使用 Typesense 实现负载分布，并确保关键的搜索基础设施能自适应负载变化。在实际使用中 Typesense 表现良好，因此我们将其移动到试验环中。

63. Vite

试验

Vite 是一款前端构建工具，在之前的技术雷达被评为评估级别后变得更加成熟与流行。它迅速地成为了我们许多团队开始前端项目时的默认选择。Vite 提供了一套基于浏览器内 ES 模块的应用的构建，打包和依赖管理工具。因为它利用了 esbuild 原生的速度和 Rollup 进行打包，Vite 显著地提升了前端开发体验。此外，当与 React 一起使用时，Vite 为广泛使用却缺乏维护的 Create React App 提供了一个有吸引力的替代品。Vite 依赖于 ES 模块，不同于大多数旧工具，它不提供 shim 和 polyfills，这表示它不兼容那些不支持 ES 模块的旧浏览器。如果要支持旧浏览器，我们的某些团队会在模块层导入 polyfills 来让 Vite 能在多个环境都能使用。

64. axe Linter

评估

对于开发者来说，在开发过程的早期发现无障碍问题变得越来越容易。像 axe-core 这样的工具可以在流水线中扫描代码来寻找无障碍问题，而 VSCode 扩展工具 axe Linter 甚至可以在这之前，例如在编写代码时就能发现它们。绝大部分无障碍问题都属于可以通过自动化测试和使用上述提到的实时反馈的提示工具（linter）来预防的类别。

65. ChatGPT

评估

ChatGPT 是一个有趣的工具，它具有在软件开发的各个方面发挥作用的潜力。作为一个已经“阅读”了数十亿个网页的大型语言模型（LLM），ChatGPT 可以提供额外的视角，协助完成不同的任务，包括生成创意和需求、创建代码和测试等。它是一种多功能的工具，能够跨越软件生命周期的多个阶段，提高开发效率并减少错误。GPT-4，作为驱动 ChatGPT 的 LLM，现在也具备与外部工具集成的能力，如知识管理库、沙盒式编码环境以及网络搜索。目前，我们认为 ChatGPT 更适合作为流程的输入，如帮助完成用户故事的初稿或编码任务的模板，而不是一个能够产出“完美周全”结果的工具。

使用这些人工智能工具可能会存在知识产权和数据隐私方面的担忧，包括一些尚未解决的法律问题，因此我们建议企业在使用前征求其法律团队的意见。我们的一些客户已经开始在软件生命周期的各个阶段尝试使用 ChatGPT，我们鼓励其他人探索这个工具并评估其潜在的作用。我们预计，像 GitHub Copilot 一样，ChatGPT 很快就会有一个“商业版”的产品，这可能会缓解知识产权方面的顾虑。

66. DataFusion

评估

DataFusion 是数据社区将 Rust 的性能、内存安全、并发特征用于数据处理的新探索。它与 Polars 相似，都提供了一套熟悉的 Rust 中的名为 DataFrame 的 API（包括 Python 绑定库），使用了 Apache Arrow 并提供了 SQL 支持。尽管它主要为单进程设计，但是也支持使用 Ballista 进行分布式运算。我们认为包括 Data Fusion 在内的用于数据处理的 Rust 库正在持续进化，它们值得关注和探索。

67. Deepchecks

评估

随着机器学习成为主流，模型测试、训练数据验证和生产模型性能监测这些实践的自动化也日渐成熟。这些自动检查越来越多地被集成进持续交付流水线或针对生产模型运行，以检测模型漂移和模型性能。已经出现了一些具有相同或类似功能的工具，以处理这个过程中的各个步骤（比如同样出现在本期技术雷达中的 Giskard 和 Evidently）。Deepchecks 是另一个此类工具，它是一个开源的 Python 库，提供了一组丰富的 API 以供流水线使用。它的一个独特的功能是，能够通过语言数据模块来处理表格或图像数据，该模块目前还在 alpha 版本。目前，没有一个单独的工具可以处理整个机器学习流水线中的各种测试和防护措施。我们建议在特定应用范围内评估 Deepchecks。

68. 设计令牌翻译工具

评估

设计令牌是定义设计体系中标准元素的有用机制。但是，在移动应用程序或 Web 应用等媒介上保持这些设计元素的一致性是一项愈发艰巨的任务。设计令牌翻译工具通过整理和自动化地将（在 YAML 或 JSON 中的）令牌描述转换为在指定媒介中控制渲染的代码（如 CSS、React 组件或 HTML），从而简化了这个问题。Style Dictionary 是一个广泛使用的开源示例，很好地集成了自动化构建流水线，但它也有商业替代品，例如 Specify。

69. Devbox

评估

Devbox 提供了易上手的界面，它利用 Nix 包管理器为每个项目创建可重现的开发环境。我们的团队使用它来消除开发环境中的版本和配置不匹配的问题，同时团队成员也喜欢它的易用性。Devbox 支持 shell hooks、自定义脚本和 devcontainer.json 生成，以便与 VSCode 集成。

70. Evidently

评估

Evidently 是一个开源的 Python 工具，旨在帮助构建对机器学习模型的监控，以确保它们的质量和在生产环境运行的稳定性。它可以用于模型生命周期的多个阶段：作为 notebook 中检查模型的仪表板，作为 pipeline 的一部分，或者作为部署后的监控。Evidently 特别关注模型漂移，同时也提供了模型质量检查、数据质量检查和目标漂变监测等功能。此外，它还提供了多种内置的指标、可视化图形和测试，可以轻松地放入报告、仪表板或测试驱动的 pipeline 中。

71. Giskard

评估

Giskard 是一个开源工具，旨在通过聚焦于可解释性和公平性来保证质量，帮助组织构建更加强大、更符合道德的 AI 模型。它促进技术和非技术利益相关者之间的合作，使他们可以共同评估模型，并建立基于避免偏见和其他必要质量指标的验收标准。Giskard 确保模型结果更好地与业务目标保持一致，并帮助解决生产部署前的质量问题。

72. GitHub Copilot

评估

GitHub Copilot 是一个由微软和 OpenAI 联合创建的人工智能编码助手。它根据开发者当前工作上下文，利用机器学习模型提供建议。它具有强大的 IDE 集成功能，可以根据现有的代码和编译器环境提供建议。尽管它被称为“您的人工智能结对编程程序员”，但我们不把它的工作称为“结对编程”——我们更倾向于称它为强化且上下文敏感的 Stack Overflow。当它正确地预测了开发人员将要做的事情时，它会是一个强大的可以帮助完成开发的工具。不过，像所有基于 LLM 的人工智能一样，它会试图使用一些看似合理，但不存在的 API，或者使用稍有问题的算法，导致系统故障。我们已经成功地在行、块和方法层面上生成了代码，同样成功地创建了测试和基础设施配置。有趣的是，当你命名良好时，它的工作效果最好，可以认为它鼓励写出可读性良好的代码。

人工智能工具的功能正在迅速发展，我们认为企业尝试这些工具是明智的。一些 Copilot 的销售宣称该工具对开发人员效率提升显著，但我们仍然持怀疑态度：毕竟，写代码并不是开发人员花时间的唯一事情，而且衡量开发人员的生产力是众所周知的困难问题。即便如此，Copilot 依然是一个相当划算的工具；如果它能提供任何生产力的提高，也是值得一试的。Copilot X——截至本文撰写时尚处于预览阶段——提供了额外的功能，并在软件开发流程中进行了整合。Copilot 有一个“商业版”产品，解决了知识产权问题，并且增加了在整个组织中集中管理工具的能力。我们认为这些功能至关重要，值得企业采购。

73. iamlive

评估

按照最小权限原则来创建我们想要的最小可行 AWS IAM 策略（Policy）可能需要经历一段很长的试错过程。而 iamlive 可以在很大程度上缩短这一过程，它监控机器上执行过的 AWS CLI 调用，并确定执行这些调用所需的策

略。该工具会生成一个包含了语句（Statement），动作（Actions），主体（Principals）以及资源（Resources）的策略文档，这可以为我们提供一个很好的开始。我们发现，iamlive 对创建用于提供基础架构的 CI/CD 流水线所需的策略特别有用，也减少了 IAM 角色策略不足导致 Terraform 运行失败后的反复尝试。

74. Kepler

评估

衡量能源消耗是团队减少软件碳足迹的重要步骤。云碳足迹（CCF）通过从云 API 检索的账单和使用数据估计能源消耗。Kepler 是基于 Kubernetes 的高效功率级别导出器（Kubernetes-based Efficient Power Level Exporter）的缩写。它通过使用软件计数器（RAPL，ACPI 和 nvml）来测量硬件资源的功耗，并采用基于 eBPF 的方法来将功耗归因于进程、容器和 Kubernetes Pod。然后，使用自定义的 ML 模型和 SPEC Power 基准测试数据将功率使用转换为能量估算。最后，将 Pod 级别的能量消耗报告作为 Prometheus 度量标准公开。在 Kubernetes 运行在虚拟机上的情况下，例如不使用裸机实例时，Kepler 使用 cgroups 来估计能源消耗。我们对云碳足迹有着丰富的经验，并且可以证明其有用性，但我们对 Kepler 项目的方法感到好奇。

75. Kubernetes External Secrets Operator

评估

Kubernetes External Secrets Operator（ESO）让我们可以将外部的密钥提供程序与 Kubernetes 集成。ESO 通过外部提供程序的 API 来读取密钥，并将其注入到 Kubernetes Secret 中。它还可以与众多的密钥管理工具集成，包括一些我们在往期技术雷达中介绍过的工具。我们的团队发现在使用 Kubernetes 的过程中，ESO 让我们可以使用统一的存储来管理整个项目的密钥，从而方便了密钥的使用。

76. Kubeshark

评估

Kubeshark 是一个 Kubernetes 的 API 流量监控工具，在 2022 年 11 月之前，它被称为 Mizu（日语“水”，象征着透明和流动性），与其他工具不同的是，Kubeshark 不需要安装监测工具或改动代码，它作为 DaemonSet 守护进程集在 Kubernetes 集群的节点层面上注入一个容器，并执行类似 tcpdump 的操作。我们认为 Kubeshark 是一个很有用的调试工具，因为它可以实时地监控多种协议（REST，gRPC，Kafka，AMQP 和 Redis）之间的所有 API 通信。

77. Obsidian

评估

知识管理对于技术工作者来说是至关重要的，因为我们需要不断地学习，并与最新的技术发展保持同步。最近，涌现出了一些笔记工具，如 Obsidian 和 Logseq，它们支持将笔记连接起来形成知识图谱，同时将文件以 markdown 格式存储在本地目录中，从而让用户拥有完全的所有权。这些工具帮助用户以一种灵活、非线性的方式组织和连接他们的笔记。

Obsidian 有一个丰富的社区插件库。其中特别引起我们注意的是 Canvas（类似于本地版的 Miro 或 Mural），以及 Dataview，它可以有效地将你的笔记作为数据库处理，并提供了一种查询语言来过滤、分类和提取 markdown 笔记中的数据。

78. Ory Kratos

评估

我们已经评估了 Ory Hydra 作为自托管的 OAuth2 解决方案，并收到了团队的正向反馈。这一次，我们转向 Ory Kratos，这是一个 API 优先的身份和用户管理系统，对开发人员友好，且易于定制。它已经提供了我们希望在身份管理系统中实现的常见功能，包括自助登录和注册、多因素身份验证（MFA/2FA）、账户验证和账户恢复。像 Hydra 一样，Kratos 是无头的（在希腊神话中，Hydra 意指九头蛇海德拉。即使斩断它的一颗头，也会生出新的头。——译者注），需要开发人员自己构建 UI，这给了团队更多的灵活性。开发人员还可以定制标识模式以适应不同的业务上下文。除了数据库，Kratos 没有外部依赖关系，并且很容易在不同的云环境中部署和扩展。如果您需要构建一个用户管理系统，我们建议试试 Kratos。

79. Philips 的自我托管 GitHub 运行器

评估

虽然 GitHub Actions 运行器涵盖了一系列最常见的运行时，但有时您需要一些更具体的东西来满足特定的使用场景，例如，一个不太常见的语言运行时，或者一个特定的硬件配置。这时您就需要一个自我托管的运行器。Philips 的自我托管 GitHub 运行器是一个 Terraform 模块，可以让您在 AWS EC2 Spot 实例上启动自定义运行器。当您使用自我托管运行器时，您会失去 GitHub Actions 提供的一些生命周期管理特性，而该模块则创建了一整套 Lambda 来帮助解决这类问题。这些 Lambda 为运行器的按需扩缩容做了大量工作。这有助于管理成本，并允许您缩短运行器工作时长，这在提高可重复性和安全性方面是一个好的实践。当您确实需要自我托管的运行器时，如果自己从头开始构建，您可能会缺失很多东西。请寻找类似这样的工具来代替。

语言和框架

采纳

80. Gradle Kotlin DSL
81. PyTorch

试验

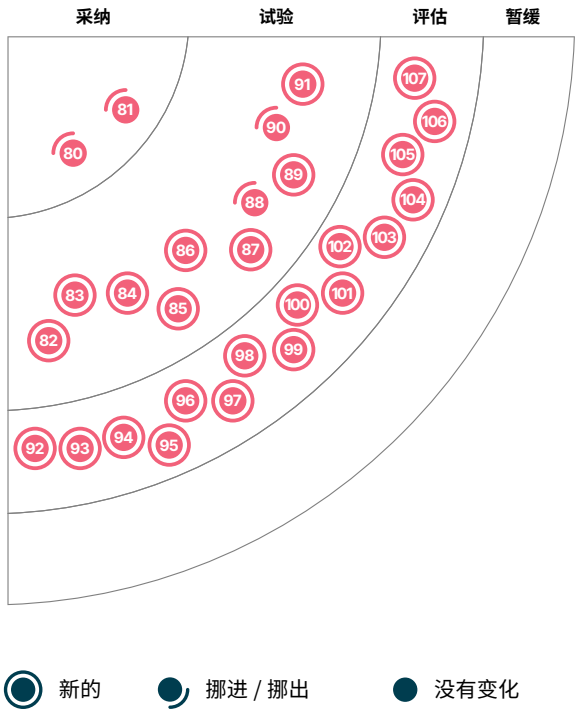
82. dbt 单元测试
83. Jetpack CameraViewfinder
84. Jetpack DataStore
85. Mikro ORM
86. 按应用设定的语言偏好设置
87. Quarto
88. River
89. Stencil
90. Synthetic Data Vault
91. Vitest

评估

92. .NET 7 Native AOT
93. .NET MAUI
94. dbt-expectations
95. Directus
96. Ferrocene
97. Flutter 嵌入式平台
98. Fugue
99. Galacean Engine
100. LangChain
101. mljar-supervised
102. nanoGPT
103. pandera
104. Qwik
105. SolidJS
106. Turborepo
107. WebXR 设备 API

Hold 暂缓

—



80. Gradle Kotlin DSL

采纳

现在，相比 Groovy，我们的团队在使用 Gradle 启动新项目时更倾向于将 Gradle Kotlin DSL（Domain-Specific Language，领域专用语言）视作默认选项。已经在使用 Groovy 的团队应考虑迁移。Kotlin 为 IDE（Integrated Development Environment，集成开发环境）中的重构与更简便的编辑提供更好的支持，而且我们的团队报告称，其产出的代码更易阅读与维护。鉴于一些 IDE 现在支持迁移，尝试替换现有的 Groovy 应该相对较快。在某些情况下，Kotlin 可能会比 Groovy 慢；然而，对于许多项目而言，这不太可能会影响到团队。

81. PyTorch

采纳

PyTorch 一直是我们选择的机器学习（ML）框架。相比于 TensorFlow，大多数团队更喜欢 PyTorch，因为它暴露了 TensorFlow 隐藏的 ML 内部工作原理，使其更易于调试。动态计算图使得模型优化比其他任何 ML 框架都更容易。State-of-the-Art（SOTA）模型的广泛可用性以及实现研究论文的便利性使 PyTorch 脱颖而出。在图 ML 领域，PyTorch Geometric 是一个更成熟的生态系统，我们的团队在使用中获得了良好的体验。PyTorch 在模型部署和扩展方面也逐渐弥合了缺失，例如，我们的团队已成功地在生产中使用 TorchServe 服务预训练模型。随着许多团队默认使用 PyTorch 来满足其端到端的深度学习需求，我们很高兴地建议采纳 PyTorch。

82. dbt 单元测试

试验

dbt 单元测试是一个 dbt 的软件包，它可以为编写数据处理模型和其逻辑的单元测试提供对依赖项的模拟。这为数据处理领域带来了更加准确的快速开发反馈。尽管它只能应用于简单的数据转换，我们的团队在 Snowflake 上依然使用该软件包实践了测试驱动开发（TDD）。虽然该软件包对失败测试的调试支持还有很大提升空间，但它依然能为编写数据转换器的单元测试提供一个优雅的开发体验。

83. Jetpack CameraViewfinder

试验

在为 Android 应用程序添加相机功能时，开发人员必须注意潜在的风险。最近推出的 Jetpack CameraViewfinder API 在显著改善了开发人员在这方面的体验。它内部使用了 TextureView 或 SurfaceView 来显示相机的视频流，并对其进行必要的转换以正确显示取景器，比如修正宽高比、缩放和旋转方向。此外，还提供了针对可折叠设备优化的布局。虽然不是一个主要的功能，但我们在这里强调它的存在，以确保团队意识到有这个选项。

84. Jetpack DataStore

试验

Jetpack DataStore 是一个新的数据存储解决方案，提供异步性、一致性和事务性的存储数据功能。它有两种实现方式：Preferences DataStore 用于无类型的键值对的存储，Proto DataStore 用于使用 Protobufs 的复杂

数据类型的存储。默认情况下，它与 Kotlin 的 coroutines 和 Flow 一起使用，但对 RXJava 2 和 3 也有额外支持。其文档建议，如果你目前正在使用 SharedPreferences，可以考虑迁移到 DataStore，我们认同这个建议。

85. Mikro ORM

试验

Mikro ORM 是一个基于 TypeScript 的对象关系映射（ORM）框架。全栈使用 TypeScript，能从浏览器到后端为开发人员提供一致的开发体验，让他们更容易编写和维护代码。另外值得注意的是，Mikro ORM 的性能非常出色，可以快速执行查询并将延迟降低到最小。虽然 Mikro ORM 提供了吸引人的功能，但仍然需提防 ORM 的常见使用问题。ORM 框架通常很复杂，而且为关系型数据存储只提供了一个“有漏洞风险”的抽象层，因此采用一个 ORM 框架前一定需要权衡利弊。

86. 按应用设定的语言偏好设置

试验

很多人都可以使用多种语言，并且会在不同的情境下使用不同的语言。运行应用程序的设备和平台通常会要求用户选择一种系统语言，然后要求应用程序也使用这个系统语言。但手机用户有时会希望某些应用能够使用与系统语言不同的语言。苹果公司早些时候在 iOS 中提供了按应用设定的语言偏好设置功能。然而在 Android 13 之前，如果安卓应用程序开发者想要提供此选项，就只能在应用程序中自行实现。Android 13 提供了一项新的系统设置：按应用设定的语言偏好设置和一个公共 API，使开发者能够更容易地提供此功能。为了向后兼容，AndroidX 还通过 AppCompatDelegate 提供了相应的 API。我们建议开发者们使用这项新功能替换之前的自定义解决方案。

87. Quarto

试验

Quarto 是一个开源的科技出版系统，它允许我们使用 markdown 方式编写文档来构建计算笔记本（notebooks），在最终的文档中插入代码及其输出。它可用于创建可重复和可定制的数据分析报告，并轻松地以各种格式进行共享。我们的数据科学团队使用 Quarto 共享包含可视化（图表）和表格的数据分析报告。他们喜欢能使用 R 和 Python 来生成这些动态报告，然后导出为 HTML 与相关人员共享。如果你希望在组织内部或外部分享你的研究和分析，我们建议评估 Quarto。

88. River

试验

许多机器学习方法的核心是从一组训练数据中创建模型。一旦创建了模型，它就可以一遍又一遍地使用。然而，世界并不是静止的，随着新数据的出现，模型往往需要更新。简单粗暴地重新训练模型可能会很慢，而且成本很高。增量学习的出现解决了这个问题，使得从数据流中不断学习成为可能，从而更快地对变化做出反应。其也有一些额外的好处：它对算力和内存的要求更低，而且可以预测。我们对 River 的实践体验仍然是正面的。Vowpal Wabbit 可以作为一个替代方案，但它的学习曲线更陡峭，而 River 提供的类似 Scikit 的 API 使 River 更容易被数据科学家所接受。

89. Stencil

试验

Stencil 是一个库，使开发人员能够使用 TypeScript、JSX 和 JSDoc 等成熟的工具构建可复用的 Web 组件。根据我们团队的经验，Stencil 是构建平台无关的设计系统的一个非常好的选择。对于少数不支持现代浏览器特性的浏览器，Stencil 通过按需 polyfilling 不支持的功能和 API 来确保兼容性。

90. Synthetic Data Vault

试验

Synthetic Data Vault（SDV）是一个数据生成工具库，它通过学习数据集的分布，生成与源数据具有相同格式和统计属性的合成数据。在往期的技术雷达中，我们讨论过应用测试环境中的生产数据的弊端。然而，生产环境中数据分布的细微差别很难手工复制，这会导致合成数据中的一些缺陷和无法预知的情况。我们使用 SDV 来生成大数据进行性能测试时获得了很好的体验。SDV 在为单表建模时表现良好。然而，随着带有外键约束的表数量的增加，数据生成时间也大大增加。尽管如此，SDV 为本地性能测试提供了很大的希望。它是一个生成合成数据的好工具，值得考虑用于你的测试需求。

91. Vitest

试验

Vitest 是一款 JavaScript 的单元测试框架。迄今为止，许多团队一直依赖 Jest，但 Jest 与现代前端构建工具 Vite 不兼容。同时使用 Jest 和 Vite 强制团队创建两个流程管道，一个用于构建和开发，另一个用于单元测试，这需要繁琐的管道配置和重复的设置。这些问题都可以通过 Vitest 解决。它专门为 Vite 设计，并使用 Vite 作为打包工具。此外，Vitest 还具有 Jest 兼容的 API，这使得在各种构建设置中可以使用 Vitest 作为 Jest 的替代品。Vite 和 Vitest 结合使用提供了更好的开发者体验，Vitest 也很快，但以我们的实际经验来看，它未必比使用 Jest 更快。

92. .NET 7 Native AOT

评估

.NET 7 Native AOT 在一众原生部署 .NET 应用程序的方法中迈出了一大步。它完全摒弃了运行时的中间语言（IL）和实时编译（JIT）。这项在 .NET 7 中引入的改进，对于在无服务器函数（Serverless Functions）中运行 .NET 应用程序意义重大。这种新的部署方式解决了一直以来在 AWS Lambda 或 Azure Functions 等无服务器平台上运行 .Net 程序的冷启动问题。相比其它部署方法，使用 Native AOT，可以生成更小的可部署二进制文件，从而缩短冷启动所需的时间。AWS 已通过 Amazon Lambda Tools 正式支持 Native AOT。这种新的部署方式令 .NET 7 的冷启动时间降低到与 TypeScript/JavaScript 一致的水平，使它成为大规模采用 .NET 基础架构的组织的可用部署方式。

93. .NET MAUI

评估

.NET MAUI 是一个使用 C# 和 XAML 创建原生移动端和桌面端应用程序的新的跨平台框架。它可以用同一份代码库构建可在 Android、iOS、macOS 和 Windows 上运行的应用程序。然而，作为一项新技术，MAUI 的生态系统仍不如 React Native 或其他跨系统平台的生态发达，并且它只支持 C#。此外，企业和组织在使用 MAUI 时可能也会面临过去使用 Xamarin 时遇到的挑战，包括糟糕的跨平台工具、移动端集成问题、开发人员难以招聘和不成熟的生态系统。

尽管微软宣称他们致力于将 MAUI 打造为以移动端为主的开源移动开发框架，但 MAUI 尚未得到市场认可。如果您已经使用了 Xamarin，可以尝试将 MAUI 作为潜在的技术升级方向进行评估。然而，如果 C# 或 Xamarin 目前不是您技术栈的一部分，在其可靠性得到验证和被市场广泛采用之前，建议您谨慎对待 MAUI。

94. dbt-expectations

评估

dbt-expectations 是一个用于 dbt 的拓展包，它产生的灵感来自于 Great Expectations。数据质量是数据治理的宗旨，所以当提及自动数据治理时，在数据管道中加入内建的管控来标记异常和质量问题很重要。就像在构建流水线中的单元测试，dbt-expectations 在数据管道运行时进行断言。在 dbt 中，你可以直接编写 Great Expectations 风格的测试对你的数据仓库数据质量测试。我们的团队已经开始探索它，并强调其是有意义的。

95. Directus

评估

我们评估了 Directus 作为一种无头内容管理系统（content management system，CMS）。尽管无头 CMS 系统有很多可选项，但我们期望它是一个自托管解决方案，包含丰富的数字资产管理和内容创作工作流程。根据这些标准，我们发现 Directus 完全满足这些需求，特别是它通过 flows 实现的事件驱动数据处理和自动化功能。

96. Ferrocene

评估

近年来 Rust 语言因其安全性、性能和并发特性而越来越受欢迎，然而，在汽车等安全攸关市场仍缺少经过认证的 Rust 工具链。目前这个缺口正在由 Ferrocene 这一 Rust 编译器工具链所填补。Ferrocene 承诺符合道路车辆电子系统的 ISO26262 功能安全标准，同时它也正在努力使该领域中使用的语言和工具链也符合条件。这部分工作也在加速推进中，我们对此感到兴奋，可以使用此类符合安全标准的工具肯定会加速 Rust 在汽车行业的应用。

97. Flutter 嵌入式平台

评估

Flutter 嵌入式平台使得创建和维护现代 UI（modern UI）变得相对容易，这种 UI 类似于移动应用程序却适用于嵌入式系统，如汽车、冰箱和其他消费类电器中的人机界面（HMI）。因为 Flutter 现在支持自定义嵌入器，从而

使其移植到不同的平台成为可能。这些应用程序使用 Dart 编程语言编写，使用 Flutter SDK 及其生态系统。我们一直在用它构建原型 – 我们的开发人员喜欢这种开发体验，我们的客户喜欢它带来的灵活、速度和现代化的用户体验。

98. Fugue

评估

在数据工程领域，可选用工具和技术数量之多，让人眼花缭乱。对于经验较少的工程师，借助一个抽象层来接触这些工具未尝不是一种合理的对策。这让他们能够专注于手头的任务，而不必分心去学习各个技术专用的 API，并且可以不费力地配置不同的底层实现技术。Fugue 就是这样一个抽象层。它为分布式计算提供统一的接口，使得在 Spark、Dask、Ray 和 DuckDB 经过较少的修改就能运行 Python、pandas 和 SQL 代码。不过，如果团队已经决定采用某一组确定的技术，并且熟悉这些技术的 API，能够对工具的系统底层做深入的调整和设置，那么这样的抽象层提供的价值就会降低。

99. Galacean Engine

评估

Galacean Engine 是一个 Web 和移动优先的交互引擎，旨在提供一种无缝的方式，以移动友好的方式渲染基于组件的架构和动画。凭借其对轻量级和高性能渲染的关注，它已成为开发者创建引人入胜的移动游戏时一个越来越受欢迎的选择。它是一个基于 TypeScript 的引擎，开发者称其性能优于其他引擎。

100. LangChain

评估

LangChain 是一个用于构建基于大型语言模型（LLMs）应用的框架。这些模型已经引起了一场生成式人工智能在各种场景下的竞赛。但是，单独使用这些 LLMs 可能是不够的——你必须将其与差异化的资产相结合去构建有影响力的产品。LangChain 提供了一些方便的功能去填补了模型和应用之间的裂痕，这些功能包括提示管理，组件链式连接，生成增强数据及丰富的用于确定执行动作和顺序的代理。我们期待基于 LLMs 演变出更多的工具和框架，并且我们推荐对 LangChain 进行评估。

101. mljar-supervised

评估

mljar-supervised 是一个 AutoML Python 包，协助理解和解释表格式数据。我们的数据科学团队很喜欢它，并已开始使用它进行自动探索性数据分析。为了找到最佳模型它抽象了如下常用方法：预处理数据、构建机器学习（ML）模型和执行超参数调整。可解释性和透明度是重要的原则，而这正是 mljar-supervised 的亮点。它会为每个 ML 模型都生成一个详细的 Markdown 报告，让你清晰地看到 ML 管道是如何构建的。这绝对是一个有趣的 AutoML 包，值得你在 ML 需求中尝试。

102. nanoGPT

评估

nanoGPT 是一个用于对中等规模的生成式预训练 Transformer（GPT）进行训练和调优的框架。其作者 Andrej Karpathy 基于注意力机制和 OpenAI 的 GPT-3 两篇论文的理论，使用 PyTorch 从零开始构建一个 GPT。在生成式人工智能火热的趋势下，我们想要强调 nanoGPT 的简洁性，并且注重对 GPT 架构的构建模块进行清晰呈现。

103. pandera

评估

在之前的雷达中，我们介绍了例如 Great Expectations 的数据验证和测试平台，其可用于验证假设并测试用于培训或分类的输入数据的质量。但是有时候，只需要一个简单的代码库就可以直接在流水线中实现测试和质量检查。pandera 是一个 Python 库，用于测试和验证跨各种框架类型的数据，例如 pandas，Dask 或者 PySpark。pandera 可以实现关于字段的简单断言或基于统计模型的假设验证。其广泛支持的框架库意味着只需编写一次测试就可以应用于各种底层数据格式。此外，pandera 还可以用于生成测试 ML 模型的合成数据 synthetic data to test ML models.

104. Qwik

评估

创建一个丰富、交互式的基于浏览器的体验的挑战之一就是尽量缩短从第一次请求到用户可以进行完整交互的时间。在网页刚打开时，应用程序可能需要下载大量的 JavaScript 到浏览器中，或者在服务器上执行一个漫长的过程来恢复应用程序状态。Qwik 是一个新的前端框架，它通过序列化应用程序状态，使其可以在服务端渲染而无需重新激活和重现程序的状态。这是通过“可恢复性”来实现的，应用程序的执行可以在服务器上暂停，并在需要时在客户端上恢复。与其他较新的前端框架（如 Astro 或 Svelte）一样，Qwik 也通过减少需要加载的 JavaScript 的数量来加速初始页面的加载。但 Qwik 在应用程序初始化时会只下载基本 HTML，大多数 JavaScript 在需要时动态地从本地缓存中加载（如果可能的话）。

105. SolidJS

评估

SolidJS 是一个用于创建用户界面的声明式 JavaScript 库。在过去的一年里，我们看到 SolidJS 在开发者中的出现次数和受欢迎程度有所提高，尤其是那些对创建更丰富的用户互动感兴趣的开发者。SolidJS 将其模板编译为真实的 DOM 节点（而不是使用虚拟 DOM），并通过细粒度的反应来更新它们，这减少了不必要的 DOM 更新，从而获得更快的性能和更好的用户体验。它的 API 很简单，并且很好的支持了 TypeScript，这对开发过程中发现错误很有帮助。SolidJS 的另一个好处是它的组件包很小，这对于快速构建和轻量级的网络应用来说非常合适，特别是移动优先的场景。SolidJS 是一个相对较新的框架，它不像其他框架那样拥有庞大的社区或生态系统。然而，从越来越多的有用的库和工具来看，它似乎越来越受欢迎。它的反应式更新系统、功能性组件模型和模板系统使 SolidJS 成为一个有吸引力的选择，我们看到一些团队和社区对其感兴趣。

106. Turborepo

评估

单一代码库的话题似乎永远会引起我们的兴趣。一些地方已经采用了单一代码库来管理整个组织的代码，而另一些地方则将这个概念应用于某些特定的应用程序，比如移动应用程序或组合的 UI/BFF 开发。无论单一代码库是否适合使用，行业似乎都在重新寻找能够有效地管理大型代码库并将它们构建成可部署单元的工具。Turborepo 是这个领域中的一个相对较新的工具，它为大型的 JavaScript 或 TypeScript 代码库提供了一种与 Nx 或 Lerna 不同的选择。大型代码库的挑战之一是如何快速地执行构建，以使其不会中断开发者的流程或降低效率。Turborepo 是用 Rust 编写的，这使得它有不错的性能；它还采用递增构建和缓存中间步骤的方法，进一步加速构建过程。但是，它会改变开发者工作流程，需要时间学习，并且最适用于具有需要多个不同的方法独立构建的大型代码库。我们发现 Turborepo 的文档很少，这导致一些团队目前仍然坚持使用更成熟的工具。然而，Turborepo 及其新的伴侣 Turbopack（目前处于测试版）是否继续发展仍值得评估和查看。

107. WebXR 设备 API

评估

在使用具有实验性的 WebVR API 时，明显可以发现将 VR 和 AR 合并到一个 API 中更加合理。为此，诞生了一个新的规范：WebXR，以避免大幅改变原有的 WebVR API。WebXR 的核心是 WebXR 设备 API，它为在网页浏览器中编写 VR 和 AR 应用提供了关键的能力。这个 API 用途广泛，但在写下该评论时，它还未被所有的浏览器完全支持。我们的团队在一些场景中使用了 WebXR，并在其中发现了 Immersive Web Working Group 所描述的好处。对于原型，我们特别喜欢的是用户可以立即在网页浏览器中进行体验。开发团队不需要到应用商店完成上架流程，用户也可以无需安装应用就可以尝试体验。考虑到 API 的现状以及它在某些浏览器中还是一个需要手动打开的隐藏功能的实际情况，我们还没有发现它在概念验证和原型之外的用途。

想要了解技术雷达最新的新闻和洞见?

请选择你喜欢的渠道来关注我们

现在订阅

Thoughtworks 是一家全球性软件及技术咨询公司，集战略、设计和工程技术咨询服务于一体。我们在 18 个国家 / 地区的 50 个办公室拥有超过 12,500 名员工。我们拥有专业卓越的跨职能团队，汇集了大量战略专家、开发人员、数据工程师和设计师，25 年多来，我们为全球各地的众多合作伙伴倾力服务，与客户一起创造了非凡的影响力，帮助他们以技术为优势解决复杂的业务问题。

/thoughtworks