智能体简介 (Introduction to Agents)

作者：Alan Blount、Antonio Gulli、Shubham Saboo、Michael Zimmermann 和 Vladimir Vuskovic

致谢 (Acknowledgements)

内容贡献者 (Content contributors)

Enrique Chan

Mike Clark

Derek Egan

Anant Nawalgaria

Kanchana Patlolla

Julia Wiesinger

策划与编辑 (Curators and editors)

Anant Nawalgaria

Kanchana Patlolla

设计师 (Designer)

Michael Lanning

智能体是语言模型向实用软件演进的自然产物。

从预测式 AI 到自主智能体 (From Predictive AI to Autonomous Agents)

人工智能正在经历深刻变革。多年来，业界关注的焦点一直是擅长处理被动式离散任务的模型——回答问题、翻译文本、根据提示生成图像。这种范式虽然强大，但每一步都离不开人工指导。

我们要看到的范式转变，是从仅仅预测或创造内容的 AI，转向一类能够自主解决问题和执行任务的新型软件。

这一新领域建立在 **AI 智能体 (AI Agents)** 之上。智能体不仅仅是静态工作流中的 AI 模型；它是一个完整的应用程序，能够制定计划并采取行动以实现目标。它结合了语言模型（Language Model, LM）的推理能力和实际行动能力，使其能够处理模型本身无法处理的复杂、多步骤任务。

关键能力在于，智能体可以自主工作，找出实现目标所需的后续步骤，而无需人工时刻介入指导。

本文是五篇系列文档的第一篇，旨在为开发人员、架构师和产品负责人提供正式指南，帮助大家从概念验证过渡到稳健的、生产级智能体系统。虽然构建简单的原型很简单，但确保安全性、质量和可靠性是一个重大挑战。本文提供了全面的基础：

- **核心解剖 (Core Anatomy)**：将智能体解构为其三个基本组件：推理模型 (Model)、可执行工具 (Tools) 和编排层 (Orchestration Layer)。
- **能力分类 (A Taxonomy of Capabilities)**：从简单的联网问题解决者到复杂的协作多智能体系统，对智能体进行分类。
- **架构设计 (Architectural Design)**：深入探讨每个组件的实际设计注意事项，从模型选择到工具实现。
- **生产环境构建 (Building for Production)**：建立评估、调试、保护和扩展智能体系统所需的智能体运维 (Agent Ops) 规范，从单个实例扩展到具有企业治理能力的机群。

在之前的《Agents 白皮书》(Agents whitepaper) 和《Agent 指南》(Agent Companion) 的基础上，本指南提供了您成功构建、部署和管理新一代智能应用程序所需的基础概念和战略框架，这些应用程序可以推理、行动和观察以完成目标。

语言已不足以描述人类与 AI 的交互方式。我们往往会将其拟人化，使用像“思考”、“推理”和“知道”这样的人类术语。我们尚缺乏词汇来区分“基于语义理解的认知”与“基于奖励函数最大化的高概率预测”。这是两种截然不同的“知道”，但在 99.X% 的情况下结果却是相同的。

AI 智能体简介 (Introduction to AI Agents)

简而言之，AI 智能体可以定义为模型、工具、编排层和运行时服务的组合，它在循环中使用 LM 来完成目标。

这四个要素构成了任何自主系统的基本架构：

- **模型 (The Model - “大脑”)**：核心语言模型 (LM) 或基础模型，作为智能体的中央推理引擎来处理信息、评估选项并做出决策。模型的类型（通用、微调或多模态）决定了智能体的认知能力。智能体系统是 LM 输入上下文窗口的最终管理者。
- **工具 (The Tools - “双手”)**：这些机制将智能体的推理与外部世界连接起来，使其行动不仅仅局限于文本生成。它们包括 API 扩展、代码函数和数据存储（如数据库或向量存储），用于访问实时的、事实性的信息。智能体系统允许 LM 计划使用哪些工具，执行工具，并将工具结果放入下一个 LM 调用的输入上下文窗口中。
- **编排层 (The Orchestration Layer - “神经系统”)**：管理智能体运行循环的控制过程。它处理计划、记忆（状态）和推理策略的执行。该层借助提示框架和推理技术（如思维链 (Chain-of-Thought) 或 ReAct (Reasoning and Acting)）将复杂目标分解为可执行步骤，并决定何时进行推理、何时调用工具。该层还负责赋予智能体”记忆”能力。
- **部署 (Deployment - “躯干和双腿”)**：虽然在笔记本电脑上构建智能体对于原型设计很有效，但生产部署才是使其成为可靠且可访问服务的关键。这涉及到将智能体托管在安全、可扩展的服务器上，并将其与监控、日志记录和管理等必要的生产服务集成。一旦部署，用户可以通过图形界面访问智能体，或者其他智能体可以通过智能体间 (Agent-to-Agent, A2A) API 以编程方式访问。

归根结底，构建生成式 AI 智能体是一种开发解决方案以解决任务的新方式。传统的开发者就像“砖瓦匠”，精确定义每一个逻辑步骤。相比之下，智能体开发者更像是一位导演。你无需为每个操作显式编写代码，而是设置场景（指导说明和提示）、选择演员（工具和 API）、并提供必要的语境（数据）。主要任务变成了指导这个自主的“演员”来实现预期的效果。

你很快就会发现，LM 最大的优势——其惊人的灵活性——也是你最大的头痛来源。大语言模型无所不能的能力，使得很难强迫它可靠且完美地只做一件特定的事情。我们过去称之为“提示工程 (Prompt Engineering)”，现在称之为“上下文工程 (Context Engineering)”，它指导 LM 生成预期的输出。对于 LM 的任何单次调用，我们输入指令、事实、可用的工具、示例、会话历史、用户画像等——用恰到好处的信息填充上下文窗口，以获得我们需要的输出。智能体就是管理 LM 输入以完成工作的软件。

当问题出现时，调试变得至关重要。“智能体运维 (Agent Ops)”本质上重新定义了测量、分析和系统优化的熟悉周期。通过追踪和日志，你可以监控智能体的“思维过程”，以识别偏离预期执行路径的情况。随着模型的发展和框架的改进，开发者的角色是提供关键组件：领域专业知识、定义的个性，以及与完成实际任务所需工具的无缝集成。

值得一提的是，全面的评估和考核往往比初始提示的影响更大。

当一个智能体被精确配置了清晰的指令、可靠的工具、作为记忆的集成上下文、优秀的用户界面、计划和解决问题的能力以及通用的世界知识时，它就超越了仅仅是“工作流自动化”的概念。它开始作为一个协作实体发挥作用：你团队中一个高效、独特适应性强且能力卓越的新成员。

本质上，智能体是一个专注于上下文窗口管理的系统。它是一个不断的循环：组装上下文、提示模型、观察结果，然后为下一步重新组装上下文。上下文可能包括系统指令、用户输入、会话历史、长期记忆、来自权威来源的基础知识、可以使用的工具以及已经调用的工具的结果。

正是这种对模型注意力的精细管理，使其推理能力得以应对新情况、达成目标。

智能体问题解决流程 (The Agentic Problem-Solving Process)

我们已经将 AI 智能体定义为一个完整的、面向目标的应用程序，它集成了推理模型、可执行工具和管理编排层。简而言之，就是“在循环中使用工具来完成目标的 LM”。

但这个系统实际上是如何工作的？从收到请求到交付结果，智能体做了什么？

在其核心，智能体在一个连续的、循环的过程中运行以实现其目标。虽然这个循环可能变得非常复杂，但它可以分解为五个基本步骤，正如《Agentic System Design》一书所详述：

1. **获取任务 (Get the Mission)**：该过程由一个特定的、高级别的目标启动。这个任务由用户提供（例如，“组织我的团队即将到来的会议差旅”）或由自动触发器提供（例如，“一个新的高优先级客户工单已到达”）。
2. **扫描场景 (Scan the Scene)**：智能体感知其环境以收集上下文。这涉及编排层访问其可用资源：“用户的请求说什么?”，“我的短期记忆中有什么信息？我之前尝试过做这个任务吗？用户上周给过我指导吗?”，“我可以从我的工具（如日历、数据库或 API）中访问什么?”
3. **深思熟虑 (Think It Through)**：这是智能体的核心“思考”循环，由推理模型驱动。智能体根据场景（步骤 2）分析任务（步骤 1）并制定计划。这通常是一个完整的推理链，而非单一的想法：“要预订旅行，我首先需要知道团队中有谁。我将使用 `get_team_roster` 工具。然后我需要通过 `calendar_api` 检查他们的空闲时间。”
4. **采取行动 (Take Action)**：编排层执行计划的第一个具体步骤。它选择并调用适当的工具——调用 API、运行代码函数或查询数据库。这是智能体在其内部推理之外对世界采取的行动。
5. **观察并迭代 (Observe and Iterate)**：智能体观察其行动的结果。`get_team_roster` 工具返回一个包含五个名字的列表。这个新信息被添加到智能体的上下文或“记忆”中。循环然后重复，返回步骤 3：“既然我有了名单，我的下一步是检查这五个人的日历。我将使用 `calendar_api`。”

这种“思考、行动、观察”循环持续进行——由编排层管理，由模型推理，由工具执行，直到智能体的内部计划完成且初始任务达成。

让我们看一个现实世界的例子，看看客户支持智能体是如何在这个 5 步循环中运作的：

假设用户问：“我的订单 #12345 在哪里?”

智能体会首先进入”深思熟虑”阶段，制定一个完整的策略，而非立即采取行动。它推理道：

“用户想要交货状态。为了提供完整的答案，我需要一个多步骤计划：

1. **识别 (Identify)**：我必须首先在我们的内部数据库中找到订单，确认它存在并获取详细信息。
2. **追踪 (Track)**：根据这些详细信息，我需要提取承运商的追踪号码。然后，我将查询外部承运商的 API 以获取实时状态。
3. **报告 (Report)**：最后，我必须将我收集的信息综合成一个清晰、有用的回复给用户。”

有了这个多步骤计划，智能体开始执行。

在第一个“行动”阶段，它执行计划的第一步，调用 `find_order("12345")` 工具。它观察结果——一个完整的订单记录，包括追踪号码“ZYX987”。

智能体的编排层识别出计划的第一部分已完成,并立即进入第二部分。它通过调用 `get_shipping_status("ZYX987")` 工具采取行动。它观察到新结果：“正在派送 (Out for Delivery)”。

最后，成功执行了数据收集阶段后，智能体移动到“报告”步骤。它感知到拥有所有必要的组件，计划最终消息，并通过生成回复来采取行动：“您的订单 #12345 ‘正在派送’!”

智能体系统分类体系 (A Taxonomy of Agentic Systems)

理解 5 步操作循环是拼图的第一部分。第二部分是认识到这个循环可以在复杂性上进行扩展，从而创建不同类别的智能体。

对于架构师或产品负责人来说，一个关键的初始决定是确定构建什么样的智能体。

我们可以将智能体系统分为几个广泛的级别，每一级都建立在上一级的能力之上。

Level 0: 核心推理系统 (The Core Reasoning System)

在我们拥有智能体之前，我们必须从最基本形式的“大脑”开始：推理引擎本身。在这种配置中，语言模型 (LM) 独立运行，仅根据其庞大的预训练知识进行响应，没有任何工具、记忆或与实时环境的交互。

它的优势在于这种广泛的训练，使其能够解释既定概念并深入规划如何解决问题。权衡之处在于完全没有实时感知能力；它对训练数据范围之外的任何事件或事实都是”盲目”的。

例如，它可以解释职业棒球的规则和纽约扬基队的完整历史。但如果你问，“昨晚扬基队比赛的最终比分是多少?”，它将无法回答。那场比赛是收集训练数据之后发生的特定现实世界事件，因此该信息根本不存在于其知识中。

Level 1: 联网的问题解决者 (The Connected Problem-Solver)

在这个级别，推理引擎通过连接和利用外部工具（我们架构中的”双手”组件）成为功能性智能体。其问题解决不再局限于其静态的预训练知识。

使用 5 步循环，智能体现在可以回答我们之前的问题。给定“任务”：“昨晚扬基队比赛的最终比分是多少?”，其“思考”步骤将其识别为实时数据需求。其“行动”步骤随后调用一个工具，如带有适当日期和搜索词的 Google Search API。它“观察”搜索结果（例如，“扬基队 5-3 获胜”），并将该事实综合成最终答案。

这种与世界交互的基本能力——无论是使用搜索工具查询比分、使用金融 API 查询实时股价，还是通过检索增强生成 (RAG) 查询数据库——是 Level 1 智能体的核心能力。

Level 2: 战略问题解决者 (The Strategic Problem-Solver)

Level 2 标志着能力的显著扩展，从执行简单任务转向战略性地规划复杂、多部分的目标。这里出现的关键技能是上下文工程：智能体能够主动选择、打包和管理每一步计划中最相关的信息。

智能体的准确性取决于聚焦、高质量的上下文。上下文工程精心管理模型有限的注意力，以防止过载并确保高效性能。

例如，考虑“任务”：“在位于 Mountain View 1600 Amphitheatre Parkway 的我的办公室和位于 San Francisco 1 Market St 的客户办公室之间，找一家好的咖啡店。”

Level 2 智能体将开始创建一个计划：

1. **思考**：“我必须首先找到中间点。”
    - **行动**：使用两个地址调用 Maps 工具。
    - **观察**：“中间点是 Millbrae, CA。”
2. **思考**：“现在我必须在 Millbrae 找到咖啡店。用户要求‘好的’，所以我将搜索 4 星或更高评分的地方。”

```
*   **行动**：调用 `google_places` 工具，查询 `query="coffee shop in Millbrae, CA", min_rating=4.
*   **观察**：“搜索返回 'Millbrae Coffee' 和 'The Daily Grind'。”
```

3. **思考**："我将综合这些结果并将其呈现给用户。"

这种战略规划还实现了主动协助，例如智能体阅读一封长长的航班确认邮件，提取关键上下文（航班号、日期），并通过将其添加到你的日历来采取行动。

Level 3: 协作式多智能体系统 (The Collaborative Multi-Agent System)

在最高级别，范式完全转变。我们不再构建单一的、全能的"超级智能体"，而是转向协同工作的"专家团队"，这种模式直接反映了人类组织。系统的集体力量在于这种分工。

在这里，智能体将其他智能体视为工具。想象一个"项目经理"智能体接收到一个"任务"："发布我们新的'Solaris'耳机。"

项目经理智能体不会自己做所有工作。它通过为它的专业智能体团队创建新任务来采取行动，就像现实生活中那样：

1. **委托给市场研究智能体 (MarketResearchAgent)**："分析竞争对手对降噪耳机的定价。明天之前返回一份摘要文档。"
2. **委托给营销智能体 (MarketingAgent)**："使用'Solaris'产品规格表作为上下文，起草三个版本的新闻稿。"
3. **委托给 Web 开发智能体 (WebDevAgent)**："根据附带的设计模型生成新的产品页面 HTML。"

这种协作模式虽然目前受限于当今 LM 的推理限制，但代表了从头到尾自动化整个复杂业务工作流的前沿。

Level 4: 自我进化系统 (The Self-Evolving System)

Level 4 代表了从委托到自主创造和适应的深刻飞跃。在这个级别，智能体系统可以识别自身能力的差距，并动态创建新工具甚至新智能体来填补这些差距。它从使用固定的资源集转向主动扩展它们。

沿用我们的例子，负责'Solaris'发布的"项目经理"智能体可能会意识到它需要监控社交媒体情绪，但其团队中没有这样的工具或智能体。

1. **思考 (元推理 Meta-Reasoning)**："我必须跟踪'Solaris'的社交媒体热度，但我缺乏这种能力。"
2. **行动 (自主创造 Autonomous Creation)**：它不通过失败，而是调用一个高级的 `AgentCreator` 工具，赋予其新任务："构建一个新的智能体，监控社交媒体上关键字'Solaris headphones'，执行情感分析，并报告每日摘要。"
3. **观察**：一个新的、专门的 `SentimentAnalysisAgent` 被即时创建、测试并添加到团队中，准备为原始任务做出贡献。

这种自主性，即系统可以动态扩展自身能力，将智能体团队转变为一个真正的学习型和进化型组织。

核心智能体架构：模型、工具和编排 (Core Agent Architecture: Model, Tools, and Orchestration)

我们知道智能体做什么以及它是如何扩展的。但我们实际上如何构建它？

从概念到代码的转变在于其三个核心组件的具体架构设计。

模型：AI 智能体的“大脑”(Model: The “Brain” of your AI Agent)

LM 是智能体的推理核心，其选择是一个关键的架构决策，决定了智能体的认知能力、运营成本和速度。

然而，仅仅选择基准测试得分最高的模型，往往会适得其反。智能体在生产环境中的成功很少由通用的学术基准决定。

现实世界的成功需要一个在智能体基础方面表现出色的模型：卓越的推理能力以应对复杂、多步骤的问题，以及可靠的工具使用能力以与世界交互。

为了做好这一点，首先定义业务问题，然后针对直接映射到该结果的指标测试模型。如果你的智能体需要编写代码，请在你的私有代码库上进行测试。如果是处理保险索赔，请评估其从特定文档格式中提取信息的能力。然后，必须将此分析与成本和延迟的实际情况进行交叉参考。“最好的”模型是针对你的特定任务，在质量、速度和价格的最佳交汇点上的那个。

你可能会选择不止一个模型，一个“专家团队”。杀鸡焉用牛刀。稳健的智能体架构可能会使用像 Gemini 2.5 Pro 这样的前沿模型来进行初始规划和复杂推理的繁重工作，但随后将更简单的、大批量的任务——如分类用户意图或总结文本——智能地路由到像 Gemini 2.5 Flash 这样更快、更具成本效益的模型。模型路由可能是自动的或硬编码的，但它是优化性能和成本的关键策略。

同样的原则也适用于处理不同的数据类型。虽然像 Gemini Live 模式这样的原生多模态模型提供了处理图像和音频的简化路径，另一种选择是使用像 Cloud Vision API 或 Speech-to-Text API 这样的专用工具。

在这种模式下，世界首先被转换为文本，然后传递给仅语言模型进行推理。这增加了灵活性并允许使用同类最佳的组件，但也引入了显著的复杂性。

最后，AI 领域处于持续、快速的进化状态。你今天选择的模型将在六个月内被取代。“一劳永逸”的想法行不通。

为了应对这一现实，意味着需要投资于一个灵活的运营框架——“智能体运维 (Agent Ops)”实践。通过一个强大的 CI/CD 流水线，根据你的关键业务指标持续评估新模型，你可以降低风险并加速升级，确保你的智能体始终由最好的大脑提供动力，而无需进行彻底的架构大修。

工具：AI 智能体的“双手”(Tools: The “Hands” of your AI Agent)

如果模型是智能体的大脑，那么工具就是将其推理与现实连接起来的双手。

它们允许智能体超越其静态的训练数据，检索实时信息并在世界上采取行动。稳健的工具接口是一个三部分循环：定义工具能做什么、调用它以及观察结果。

以下是智能体构建者将放入其智能体“手中”的几种主要工具类型。有关更完整的深入探讨，请参阅本系列中专注于智能体工具的白皮书。

检索信息：基于事实 (Retrieving Information: Grounding in Reality)

最基础的工具是访问最新信息的能力。检索增强生成 (RAG) 给了智能体一张“借书证”来查询外部知识，这些知识通常存储在向量数据库或知识图谱中，范围从公司内部文档到通过 Google Search (Google 搜索) 获取的网络知识。对于结构化数据，自然语言转 SQL (NL2SQL) 工具允许智能体查询数据库以回答分析性问题，如“上个季度我们最畅销的产品是什么?”通过在说话之前进行查找——无论是在文档中还是数据库中——智能体将自己建立在事实之上，大大减少了幻觉。

执行行动：改变世界 (Executing Actions: Changing the World)

当智能体从读取信息转向主动做事时，它们真正的力量才被释放出来。通过将现有的 API 和代码函数包装为工具，智能体可以发送电子邮件、安排会议或更新 ServiceNow 中的客户记录。对于更动态的任务，智能体可以即时编写和执行代码。在安全的沙箱中，它可以生成 SQL 查询或 Python 脚本来解决复杂问题或执行计算，将其从知识渊博的助手转变为自主的行动者。

这还包括用于人类交互的工具。智能体可以使用人机回环 (HITL) 工具暂停其工作流并请求确认（例如 `ask_for_confirmation()`）或从用户界面请求特定信息（例如 `ask_for_date_input()`），确保关键决策有人参与。HITL 可以通过短信和数据库中的任务来实现。

函数调用：将工具连接到你的智能体 (Function Calling: Connecting Tools to your Agent)

为了让智能体可靠地进行“函数调用”并使用工具，它需要清晰的指令、安全的连接和编排。像 OpenAPI 规范这样的长期标准提供了这一点，给智能体一份结构化的合同，描述工具的目的、所需参数和预期响应。此模式让模型每次都能生成正确的函数调用并解释 API 响应。为了更简单地发现和连接工具，像模型上下文协议 (MCP) 这样的开放标准因其更方便而变得流行。此外，少数模型拥有原生工具，如带有原生 Google Search 的 Gemini，其中函数调用作为 LM 调用本身的一部分发生。

编排层 (The Orchestration Layer)

如果模型是智能体的大脑，工具是它的双手，那么编排层就是连接它们的“中枢神经系统”。它是运行“思考、行动、观察”循环的引擎，是管理智能体行为的状态机，也是开发者的精心设计的逻辑变为现实的地方。这一层不仅仅是管道；它是整个智能体交响乐的指挥，决定模型何时应该推理，应该使用哪个工具，以及该行动的结果应如何告知下一步动作。

核心设计选择 (Core Design Choices)

第一个架构决策是确定智能体的自主程度。这个选择存在于一个频谱上。在一端，你有确定性的、可预测的工作流，它们调用 LM 作为特定任务的工具——只是一点 AI 来增强现有流程。在另一端，LM 处于驾驶座，动态地适应、规划和执行任务以实现目标。

一个并行的选择是实现方法。无代码构建器提供速度和可访问性，通过授权业务用户自动化结构化任务并快速构建简单的智能体。对于更复杂、关键任务系统，代码优先框架，如 Google 的智能体开发套件 (Agent Development Kit, ADK)，提供了工程师所需的深度控制、可定制性和集成能力。

无论采用哪种方法，生产级框架都是必不可少的。它必须是开放的，允许你插入任何模型或工具以防止供应商锁定。它必须提供精确的控制，支持混合方法，即 LM 的非确定性推理受硬编码业务规则的管辖。最重要的是，框架必须为可观测性而构建。当智能体行为异常时，你不能简单地在模型的“思想”中设置断点。一个强大的框架会生成详细的追踪和日志，通过追踪暴露整个推理轨迹：模型的内部独白、它选择的工具、它生成的参数以及它观察到的结果。

注入领域知识和个性 (Instruct with Domain Knowledge and Persona)

在这个框架内，开发者最强大的杠杆是向智能体灌输领域知识和独特的个性。这是通过系统提示或一组核心指令来完成的。这不仅仅是一个简单的命令；它是智能体的宪法。

在这里，你告诉它，“你是一个 Acme Corp 的乐于助人的客户支持智能体……”并提供约束、所需的输出模式、参与规则、特定的语气，以及关于何时以及为何应使用其工具的明确指导。指令中的几个示例场景通常非常有效。

利用上下文进行增强 (Augment with Context)

智能体的“记忆”在运行时被编排进 LM 上下文窗口。有关更完整的深入探讨，请参阅本系列中专注于智能体记忆的白皮书。

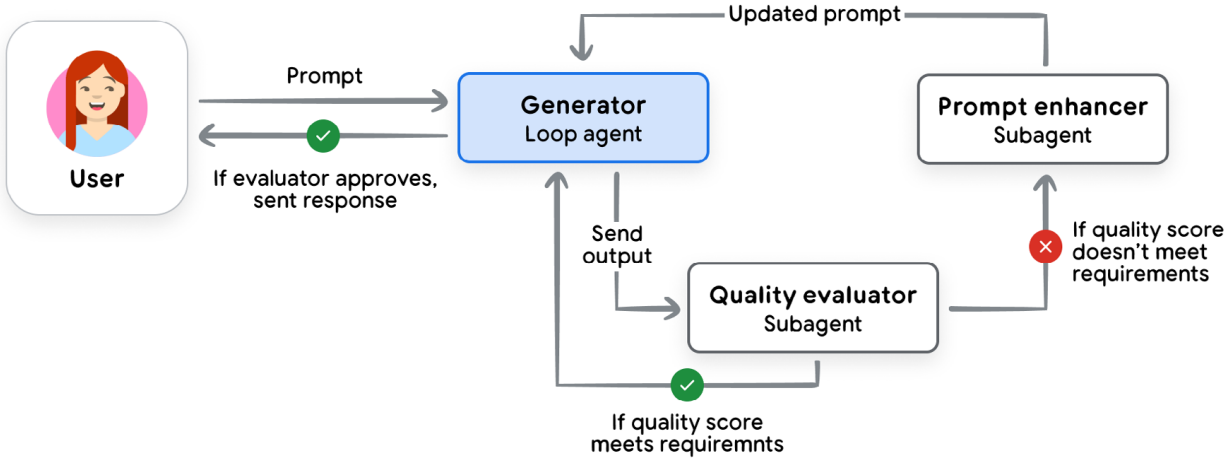
短期记忆是智能体的活跃“便笺簿”，维护当前对话的运行历史。它跟踪正在进行的循环中的（行动，观察）对序列，提供模型决定下一步做什么所需的直接上下文。这可以作为状态、工件、会话或线程等抽象来实现。

长期记忆提供跨会话的持久性。在架构上，这几乎总是作为另一个专门的工具来实现——连接到向量数据库或搜索引擎的 RAG 系统。编排器赋予智能体预取和主动查询其自身历史的能力，使其能够“记住”用户的偏好或几周前类似任务的结果，从而获得真正个性化和连续的体验。

多智能体系统与设计模式 (Multi-Agent Systems and Design Patterns)

随着任务复杂性的增加，构建单一的、全能的“超级智能体”变得效率低下。

更有效的解决方案是采用“专家团队”方法，这反映了人类组织。这是多智能体系统的核心：一个复杂的过程被细分为离散的子任务，每个子任务都分配给一个专门的 AI 智能体。



这种分工允许每个智能体更简单、更专注，并且更容易构建、测试和维护，这对于动态或长期运行的业务流程是理想的。

架构师可以依赖经过验证的智能体设计模式，尽管智能体能力以及因此模式正在迅速发展。对于动态或非线性任务，协调器 (Coordinator) 模式至关重要。它引入了一个“经理”智能体，分析复杂请求，分割主要任务，并智能地将每个子任务路由到适当的专家智能体（如研究员、作家或编码员）。协调器然后聚合来自每个专家的响应，以制定最终的、全面的答案。

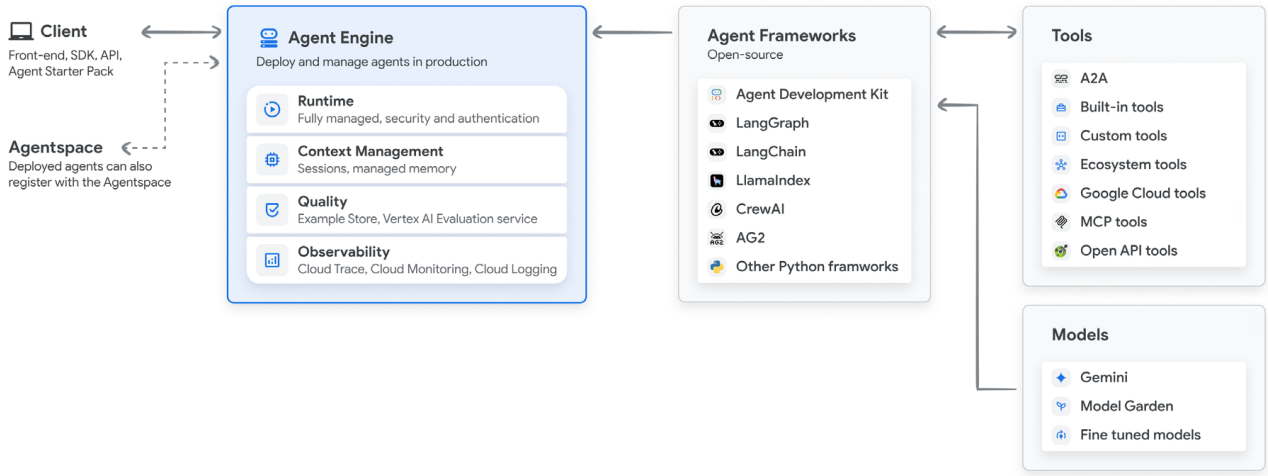
对于更线性的工作流，顺序 (Sequential) 模式更合适，就像数字装配线一样，一个智能体的输出直接成为下一个智能体的输入。其他关键模式关注质量和安全。迭代优化 (Iterative Refinement) 模式创建一个反馈循环，使用“生成器”智能体创建内容，并使用“批评家”智能体根据质量标准对其进行评估。

对于高风险任务，人机回环 (HITL) 模式至关重要，在工作流中创建一个刻意的暂停，以便在智能体采取重大行动之前获得人的批准。

智能体部署和服务 (Agent Deployment and Services)

在你构建了本地智能体之后，你会希望将其部署到服务器上，以便它一直在运行，并且其他人和其他智能体可以使用它。继续我们的类比，部署和服务将是我们智能体的躯干和双腿。智能体需要多种服务才能有效，如会话历史和记忆持久化等。作为智能体构建者，你还将负责决定记录什么，以及采取什么安全措施来保护数据隐私、数据驻留和法规遵从性。所有这些服务都在将智能体部署到生产环境的范围内。

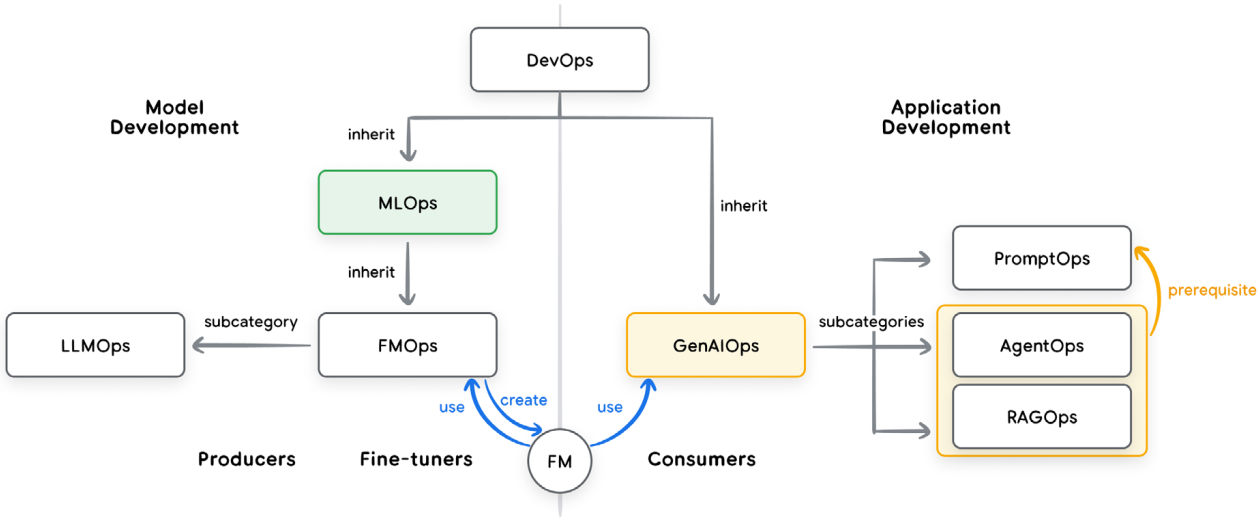
幸运的是，智能体构建者可以依赖数十年的应用程序托管基础设施。智能体毕竟是一种新形式的软件，许多相同的原则适用。构建者可以依赖专门构建的、针对智能体的部署选项，如 Vertex AI Agent Engine，它在一个平台中支持运行时和其他所有内容。对于想要更直接控制其应用程序堆栈的软件开发人员，或者在其现有 DevOps 基础设施中部署智能体的开发人员，任何智能体和大多数智能体服务都可以添加到 Docker 容器中，并部署到行业标准运行时（如 Cloud Run 或 GKE）上。



如果你不是软件开发人员和 DevOps 专家，部署你的第一个智能体的过程可能会让人望而生畏。许多智能体框架通过部署命令或专用平台使部署变得容易，这些应该用于早期探索和入职。升级到安全且生产就绪的环境通常需要投入更多时间并应用最佳实践，包括 CI/CD 和智能体的自动化测试。

智能体运维 (Agent Ops)：应对不可预测性的结构化方法 (Agent Ops: A Structured Approach to the Unpredictable)

当你构建你的第一个智能体时，你会一遍又一遍地手动测试其行为。当你添加一个功能时，它能工作吗？当你修复一个错误时，你是否引起了不同的问题？测试对于软件开发来说是正常的，但对于生成式 AI 来说，情况有所不同。



从传统的、确定性软件到随机的、智能体系统的转变需要一种新的运营哲学。传统的软件单元测试可以简单地断言 `output == expected`；但这在智能体的响应本质上是概率性的情况下不起作用。此外，因为语言很复杂，它通常需要 LM 来评估“质量”——即智能体的响应做了它应该做的一切，没有做不应该做的事，并且语气恰当。

智能体运维 (Agent Ops) 是管理这一新现实的纪律严明、结构化的方法。它是 DevOps 和 MLOps 的自然演变，专为构建、部署和治理 AI 智能体的独特挑战而量身定制，将不可预测性从负债转变为管理的、可测量的和可靠的功能。有关更完整的深入探讨，请参阅本系列中专注于智能体质量的白皮书。

衡量重要的指标：像 A/B 实验一样通过仪器检测成功 (Measure What Matters: Instrumenting Success Like an A/B Experiment)

在你改进你的智能体之前，你必须定义在你的业务背景下“更好”意味着什么。像 A/B 测试一样构建你的可观测性策略，并问自己：证明智能体正在交付价值的关键绩效指标 (KPI) 是什么？这些指标应该超越技术正确性，衡量现实世界的影响：目标完成率、用户满意度评分、任务延迟、每次交互的运营成本，以及——最重要的——对收入、转化率或客户留存率等业务目标的影响。

这种自上而下的视图将指导你其余的测试，使你走上指标驱动开发的道路，并让你计算投资回报率。

质量而非通过/失败：使用 LM 法官 (Quality Instead of Pass/Fail: Using a LM Judge)

业务指标不会告诉你智能体是否行为正确。由于简单的通过/失败是不可能的，我们转向使用“LM 作为法官”来评估质量。这涉及使用强大的模型根据预定义的标准评估智能体的输出：它给出了正确的答案吗？响应是否有事实依据？它遵循指令了吗？这种针对黄金提示数据集运行的自动化评估，提供了质量的一致性衡量标准。

创建评估数据集——包括理想（或“黄金”）问题和正确答案——可能是一个乏味的过程。为了构建这些，你应该从现有的生产或开发交互中采样场景。数据集必须覆盖你期望用户使用的全部用例，外加一些意外情况。虽然在评估上的投入很快就会得到回报，但评估结果在被接受为有效之前，应始终由领域专家审查。

这些评估的策划和维护正日益成为产品经理在领域专家支持下的关键责任。

指标驱动开发：你部署的行动/不行动信号 (Metrics-Driven Development: Your Go/No-Go for Deployment)

一旦你自动化了数十个评估场景并建立了可信的质量评分，你就可以自信地测试对开发智能体的更改。过程很简单：在整个评估数据集上运行新版本，并将评分直接与现有的生产版本进行比较。这个稳健的系统消除了猜测，确保你对每一次部署都充满信心。虽然自动化评估至关重要，但不要忘记其他重要因素，如延迟、成本和任务成功率。为了获得最大的安全性，使用 A/B 部署来缓慢推出新版本，并将这些真实的生产指标与你的模拟评分一起比较。

使用 OpenTelemetry 追踪进行调试：回答“为什么?” (Debug with OpenTelemetry Traces: Answering “Why?”)

当你的指标下降或用户报告错误时，你需要了解“为什么”。OpenTelemetry 追踪是智能体整个执行路径（轨迹）的高保真、逐步记录，允许你调试智能体的步骤。通过追踪，你可以看到发送给模型的具体提示、模型的内部推理（如果可用）、它选择调用的具体工具、它为该工具生成的精确参数，以及作为观察结果返回的原始数据。第一次看追踪可能会觉得复杂，但它们提供了诊断和修复任何问题根本原因所需的细节。重要的追踪细节可以转化为指标，但审查追踪主要用于调试，而不是性能概览。

追踪数据可以在像 Google Cloud Trace 这样的平台中无缝收集，这些平台可视化并搜索大量追踪，简化根本原因分析。

珍视人类反馈：指导你的自动化 (Cherish Human Feedback: Guiding Your Automation)

人类反馈不是要处理的烦恼；它是你拥有的用于改进智能体的最有价值和数据最丰富的资源。当用户提交错误报告或点击“拇指向下”按钮时，他们是在给你一份礼物：一个新的、现实世界的边缘情况，你的自动化评估场景错过了它。收集和聚合这些数据至关重要；当你看到统计上大量的类似报告或指标下降时，你必须将这些发生率关联回你的分析平台，以生成见解并触发运营问题的警报。一个有效的 Agent Ops 流程通过捕获此反馈、复制问题并将该特定场景转换为评估数据集中的一个新的、永久的测试用例来“闭环”。这确保你不仅修复了错误，还为系统建立了免疫机制，防止同类错误再次发生。

智能体互操作性 (Agent Interoperability)

一旦你构建了高质量的智能体，你会希望能够将它们与用户和其他智能体互连。在我们的身体部位类比中，这将是智能体的脸。连接到智能体与将智能体与数据和 API 连接是有区别的；智能体不是工具。假设你已经将工具连接到你的智能体中，现在让我们考虑如何将你的智能体带入更广泛的生态系统。

智能体与人类 (Agents and Humans)

最常见的智能体-人类交互形式是通过用户界面。在其最简单的形式中，这是一个聊天机器人，用户输入请求，作为后端服务的智能体处理它并返回一段文本。更高级的智能体可以提供结构化数据，如 JSON，以驱动丰富、动态的前端体验。人机回环 (HITL) 交互模式包括意图细化、目标扩展、确认和澄清请求。

计算机使用 (Computer use) 是一类工具，其中 LM 接管用户界面的控制权，通常带有人类交互和监督。启用计算机使用的智能体可以决定下一个最佳操作是导航到新页面、突出显示特定按钮或用相关信息预填表单。

更进一步，LM 可以直接更改 UI 以满足当下的需求。这可以通过控制 UI 的工具 (MCP UI)，或可以与智能体同步客户端状态的专用 UI 消息系统 (AG UI)，甚至生成定制界面 (A2UI) 来完成。

当然，人类交互不仅限于屏幕和键盘。高级智能体正在突破纯文本交互的局限，借助”实时模式”实现更自然的类人连接和实时多模态通信。像 Gemini Live API 这样的技术实现了双向流式传输，允许用户与智能体交谈并打断它，就像他们在自然对话中那样。

这种能力从根本上改变了智能体-人类协作的性质。通过访问设备的摄像头和麦克风，智能体可以看到用户所见，听到用户所说，并以模仿人类对话的延迟响应生成的语音。

这开启了大量用文本根本无法实现的用例，从技术人员在维修设备时接收免提指导，到购物者获得实时风格建议。它使智能体成为一个更直观和可访问的合作伙伴。

智能体与智能体 (Agents and Agents)

就像智能体必须与人类连接一样，它们也必须彼此连接。随着企业扩展其 AI 的使用，不同的团队将构建不同的专业智能体。如果没有通用标准，连接它们将需要构建一个难以维护的、纠缠不清的脆弱、自定义 API 集成网络。核心挑战是双重的：发现（我的智能体如何找到其他智能体并知道它们能做什么?）和沟通（我们如何确保它们说同一种语言?）。

Agent2Agent (A2A) 协议是旨在解决这个问题的开放标准。它充当智能体经济的通用握手。A2A 允许任何智能体发布一张数字“名片”，称为“智能体卡片 (Agent Card)”。这个简单的 JSON 文件通告了智能体的能力、其网络端点以及与之交互所需的凭证。

这使得发现变得简单和标准化。与专注于解决交易请求的 MCP 不同，Agent2Agent 通信通常用于额外的解决问题。

一旦被发现，智能体使用面向任务的架构进行通信。交互被框架化为异步”任务”，而非简单的请求-响应模式。客户端智能体向服务器智能体发送任务请求，服务器智能体随后可以在通过长时间运行的连接处理问题时提供流式更新。这种稳健的、标准化的通信协议是拼图的最后一块，实现了代表自动化前沿的协作式、Level 3 多智能体系统。A2A 将一组孤立的智能体转变为一个真正的、可互操作的生态系统。

智能体与金钱 (Agents and Money)

随着 AI 智能体为我们做更多任务，其中一些任务涉及购买或出售、谈判或促进交易。当前的 web 是为人类点击“购买”而构建的，责任在于人。如果自主智能体点击“购买”，它会引发信任危机——如果出了问题，谁的错？这些是授权、真实性和问责制的复杂问题。为了解锁真正的智能体经济，我们需要新的标准，允许智能体代表其用户安全可靠地进行交易。

这个新兴领域远未建立，但两个关键协议正在铺平道路。智能体支付协议 (Agent Payments Protocol, AP2) 是一项开放协议，旨在成为智能体商业的权威语言。它通过引入加密签名的数字“授权书 (Mandates)”扩展了像 A2A 这样的协议。这些充当用户意图的可验证证明，为每笔交易创建一个不可否认的审计跟踪。这允许智能体根据用户的授权，在全球范围内安全地浏览、谈判和交易。

与之互补的是 x402，这是一个开放的互联网支付协议，使用标准的 HTTP 402 “Payment Required”状态码。它实现了无摩擦的、机器对机器的小额支付，允许智能体按使用付费的方式支付 API 访问或数字内容费用，而无需复杂的账户或订阅。这些协议一起正在构建智能体网络的信任基础层。

保护单个智能体：信任权衡 (Securing a Single Agent: The Trust Trade-Off)

当你创建第一个 AI 智能体时，你立即面临一个基本的紧张关系：效用与安全之间的权衡。为了让智能体有用，你必须给它权力——做决定的自主权和执行行动（如发送电子邮件或查询数据库）的工具。然而，你赋予的每一分权力都会引入相应程度的风险。主要的安全问题是流氓行动——意外或有害的行为——以及敏感数据泄露。你想给智能体足够的自主空间去完成任务，但又要加以约束以防止其失控，尤其是当涉及不可逆的行动或公司的私有数据时。

为了管理这一点，你不能仅依靠 AI 模型的判断，因为它可能被提示注入等技术操纵。相反，最佳实践是混合的、纵深防御的方法。第一层由传统的、确定性的护栏组成——一组硬编码的规则，作为模型推理之外的安全瓶颈。这可能是一个策略引擎，阻止任何超过 100 美元的购买，或在智能体与外部 API 交互之前需要明确的用户确认。这一层为智能体的权力提供了可预测的、可审计的硬性限制。

第二层利用基于推理的防御，使用 AI 来帮助保护 AI。这涉及训练模型以使其更能抵御攻击（对抗性训练）并采用较小的、专门的“守卫模型”充当安全分析师。这些模型可以在执行之前检查智能体提议的计划，标记潜在的风险或违反策略的步骤以供审查。这种混合模型结合了代码的严格确定性和 AI 的语境意识，为即使是单个智能体也创建了稳健的安全态势，确保其权力始终与其目的保持一致。

智能体身份：一类新的主体 (Agent Identity: A New Class of Principal)

在传统的安全模型中，有人类用户可能使用 OAuth 或 SSO，也有使用 IAM 或服务账户的服务。智能体增加了第三类主体。智能体不仅仅是一段代码；它是一个自主的参与者，一种需要其自己可验证身份的新型主体。就像员工被发放身份证一样，平台上的每个智能体必须被发放一个安全的、可验证的“数字护照”。这个智能体身份不同于调用它的用户的身份和构建它的开发者的身份。

这是我们必须如何在企业中处理身份和访问管理 (IAM) 的根本转变。

让每个身份都被验证并拥有对所有身份的访问控制，是智能体安全的基石。一旦智能体拥有加密可验证的身份（通常使用像 SPIFFE 这样的标准），它就可以被授予其自己的特定、最小特权权限。SalesAgent 被授予对 CRM 的读/写访问权限，而 HRonboardingAgent 被明确拒绝。这种细粒度的控制至关重要。它确保即使单个智能体被破坏或行为异常，潜在的爆炸半径也被控制住。没有智能体身份构造，智能体就无法代表具有有限委托权力的用户工作。

主体实体	认证 / 验证	备注
用户	使用 OAuth 或 SSO 认证	具有完全自主权并对其行为负责的人类参与者
智能体 (新的主体类别)	使用 SPIFFE 验证	智能体拥有委托的权力，代表用户采取行动
服务账户	集成到 IAM	应用程序和容器，完全确定性，不对行动负责

限制访问的策略 (Policies to Constrain Access)

策略是一种授权 (AuthZ) 形式，不同于认证 (AuthN)。通常，策略限制主体的能力；例如，“市场部的用户只能访问这 27 个 API 端点，不能执行 DELETE 命令。”当我们开发智能体时，我们需要将权限应用于智能体、它们的工具、其他内部智能体、它们可以共享的上下文以及远程智能体。这样想：如果你将所有 API、数据、工具和智能体添加到你的系统中，那么你必须将访问限制为仅完成其工作所需的那部分功能的子集。这是推荐的方法：应用最小特权原则，同时保持上下文相关性。

保护 ADK 智能体 (Securing an ADK Agent)

随着身份和策略的核心原则确立，保护使用 Agent Development Kit (ADK) 构建的智能体成为通过代码和配置应用这些概念的实际练习。

如上所述，该过程需要清晰定义的身份：用户账户（例如 OAuth）、服务账户（运行代码）、智能体身份（使用委托权力）。

一旦处理了认证，下一层防御涉及建立策略以限制对服务的访问。这通常在 API 治理层完成，同时也支持 MCP 和 A2A 服务治理。

下一层是在你的工具、模型和子智能体中构建护栏以强制执行策略。这确保了无论 LM 推理什么或恶意提示可能建议什么，工具自身的逻辑都将拒绝执行不安全或违反策略的行动。这种方法提供了可预测和可审计的安全基线，将抽象的安全策略转化为具体的、可靠的代码。

为了实现更能适应智能体运行时行为的动态安全性，ADK 提供了回调和插件。`before_tool_callback` 允许你在工具调用运行之前检查其参数，根据智能体的当前状态验证它们，以防止不一致的行动。对于更可重用的策略，你可以构建插件。一个常见的模式是“Gemini 作为法官 (Gemini as a Judge)”，它使用像 Gemini Flash-Lite 这样快速、廉价的模型或你自己微调的 Gemma 模型来实时筛选用户输入和智能体输出，以查找提示注入或有害内容。

对于更喜欢全托管、企业级解决方案进行动态检查的组织，Model Armor 可以作为可选服务集成。Model Armor 充当专门的安全层，筛选提示和响应以查找各种威胁，包括提示注入、越狱尝试、敏感数据 (PII) 泄露和恶意 URL。通过将这些复杂的安全任务交由专用服务处理，开发人员可以确保一致、稳健的保护，而无需自己构建和维护这些护栏。这种 ADK 内的混合方法——结合强大的身份、确定性的工具内逻辑、动态的 AI 驱动护栏以及像 Model Armor 这样的可选托管服务——是你如何构建既强大又值得信赖的单一智能体的方法。

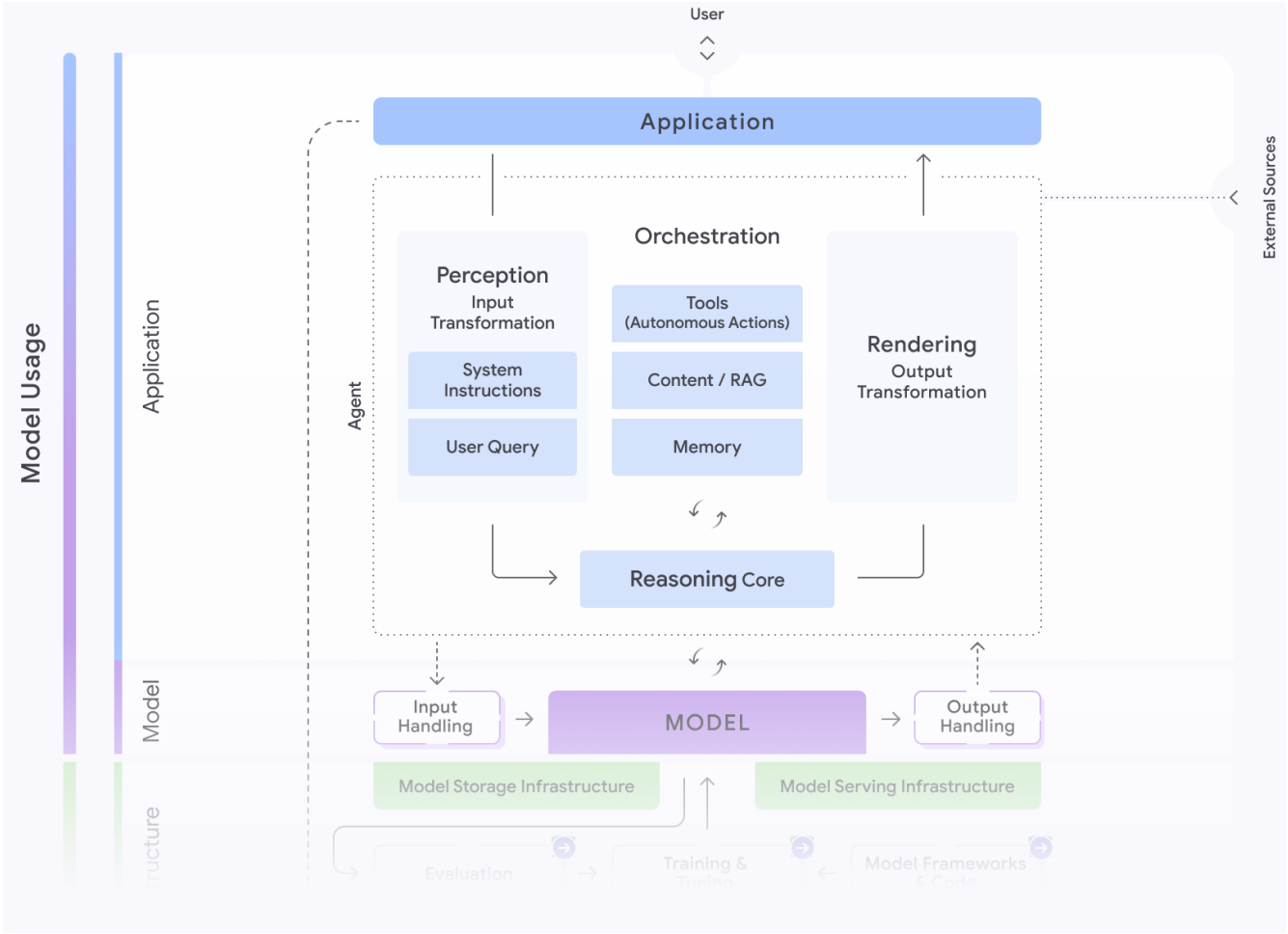


图 6：安全与智能体，来源：https://saif.google/focus-on-agents

从单一智能体扩展到企业级机群 (Scaling Up from a Single Agent to an Enterprise Fleet)

单一 AI 智能体的生产成功是一个胜利。扩展到数百个机群是一个架构挑战。如果你正在构建一两个智能体，你的关注点主要关于安全。如果你正在构建许多智能体，你必须设计系统来处理更多事情。就像 API 蔓延一样，当智能体和工具在组织内激增时，它们创造了一个新的、复杂的交互、数据流和潜在安全漏洞网络。

管理这种复杂性需要一个更高阶的治理层，集成你所有的身份和策略，并报告给中央控制平面。

安全与隐私：加固智能体前沿 (Security and Privacy: Hardening the Agentic Frontier)

企业级平台必须解决生成式 AI 固有的独特安全和隐私挑战，即使只有单个智能体在运行。智能体本身成为一个新的攻击向量。恶意行为者可以尝试提示注入以劫持智能体的指令，或数据投毒以破坏其用于训练或 RAG 的信息。此外，限制不当的智能体可能会在响应中无意泄露敏感的客户数据或专有信息。

稳健的平台提供纵深防御策略来减轻这些风险。它从数据开始，确保企业的专有信息永远不会用于训练基础模型，并受到像 VPC Service Controls 这样的控制保护。它需要输入和输出过滤，就像提示和响应的防火墙一样。最后，平台必须提供像知识产权赔偿这样的合同保护，既针对训练数据也针对生成的内容，给企业部署智能体到生产环境所需的法律和技术信心。

智能体治理：控制平面而非蔓延 (Agent Governance: A Control Plane instead of Sprawl)

随着智能体及其工具在组织内激增，它们创造了一个新的、复杂的交互和潜在漏洞网络，这是通常被称为“智能体蔓延”的挑战。管理这一点需要超越保护单个智能体，实施更高阶的架构方法：一个作为所有智能体活动控制平面的中央网关。

想象一个繁忙的大都市，成千上万的自动驾驶车辆——用户、智能体和工具——都在有目的地移动。如果没有红绿灯、车牌和中央控制系统，混乱将占据主导地位。网关方法创建了那个控制系统，为所有智能体流量建立强制入口点，包括用户对智能体的提示或 UI 交互、智能体对工具的调用 (via MCP)、智能体对智能体的协作 (via A2A) 以及直接对 LM 的推理请求。通过坐在这个关键路口，组织可以检查、路由、监控和管理每一次交互。

这个控制平面服务于两个主要、相互关联的功能：

1. **运行时策略执行 (Runtime Policy Enforcement)**：它充当实施安全的架构瓶颈。它处理认证（“我知道这个行动者是谁吗?”）和授权（“他们有权限做这个吗?”）。集中化执行为可观测性提供了“单一玻璃板”，为每笔交易创建通用日志、指标和追踪。这将不同智能体和工作流的意大利面条式代码转变为透明和可审计的系统。
2. **集中化治理 (Centralized Governance)**：为了有效地强制执行策略，网关需要一个真理来源。这由一个中央注册表——智能体和工具的企业应用商店——提供。该注册表允许开发人员发现和重用现有资产，防止重复工作，同时给管理员一个完整的清单。更重要的是，它实现了智能体和工具的正式生命周期，允许在发布之前进行安全审查、版本控制，以及创建细粒度的策略，规定哪些业务部门可以访问哪些智能体。

通过结合运行时网关和中央治理注册表，组织将混乱蔓延的风险转化为一个管理的、安全的和高效的生态系统。

成本与可靠性：基础设施基础 (Cost and Reliability: The Infrastructure Foundation)

归根结底，企业级智能体必须既可靠又具有成本效益。一个经常失败或提供缓慢结果的智能体具有负投资回报率。相反，一个昂贵得令人望而却步的智能体无法扩展以满足业务需求。底层基础设施必须设计为管理这种权衡，既安全又符合监管和数据主权合规性。

在某些情况下，当特定智能体或子功能的流量不规则时，你需要的功能是缩容至零（scale-to-zero）。对于关键任务、对延迟敏感的工作负载，平台必须提供专用的、有保证的容量，例如用于 LM 服务的预置吞吐量

(Provisioned Throughput) 或针对 Cloud Run 等运行时的 99.9% 服务水平协议 (SLA)。这提供了可预测的性能，确保你最重要的智能体即使在重负载下也能始终响应。通过提供这种基础设施选项谱系，加上对成本和性能的全面监控，你建立了将 AI 智能体从有希望的创新扩展为企业核心、可靠组件的最终必要基础。

智能体如何进化和学习 (How agents evolve and learn)

在现实世界中部署的智能体在策略、技术和数据格式不断变化的动态环境中运行。如果没有适应能力，智能体的性能将随着时间的推移而退化——这一过程通常被称为“老化”——导致效用和信任的丧失。手动更新大型智能体机群以跟上大变化的步伐是不经济且缓慢的。一个更可扩展的解决方案是设计能够自主学习和进化的智能体，以最少的工程努力在工作中提高其质量。

智能体如何学习和自我进化 (How agents learn and self evolve)

就像人类一样，智能体从经验和外部信号中学习。这个学习过程由几个信息源驱动：

- **运行时经验 (Runtime Experience)**：智能体从运行时工件（如会话日志、追踪和记忆）中学习，这些工件捕获了成功、失败、工具交互和决策轨迹。至关重要的是，这包括人机回环 (HITL) 反馈，它提供了权威的更正和指导。
- **外部信号 (External Signals)**：学习也由新的外部文档驱动，例如更新的企业策略、公共监管准则或其他智能体的批评。

这些信息随后用于优化智能体的未来行为。高级系统不仅仅是总结过去的交互，而是创建可概括的工件来指导未来的任务。最成功的适应技术分为两类：

- **增强的上下文工程 (Enhanced Context Engineering)**：系统不断完善其提示、少样本示例以及从记忆中检索的信息。通过优化为每个任务提供给 LM 的上下文，它增加了成功的可能性。
- **工具优化和创建 (Tool Optimization and Creation)**：智能体的推理可以识别其能力的差距并采取行动填补它们。这可能涉及获得新工具的访问权限、通过即时创建新工具（例如 Python 脚本），或修改现有工具（例如更新 API 模式）。

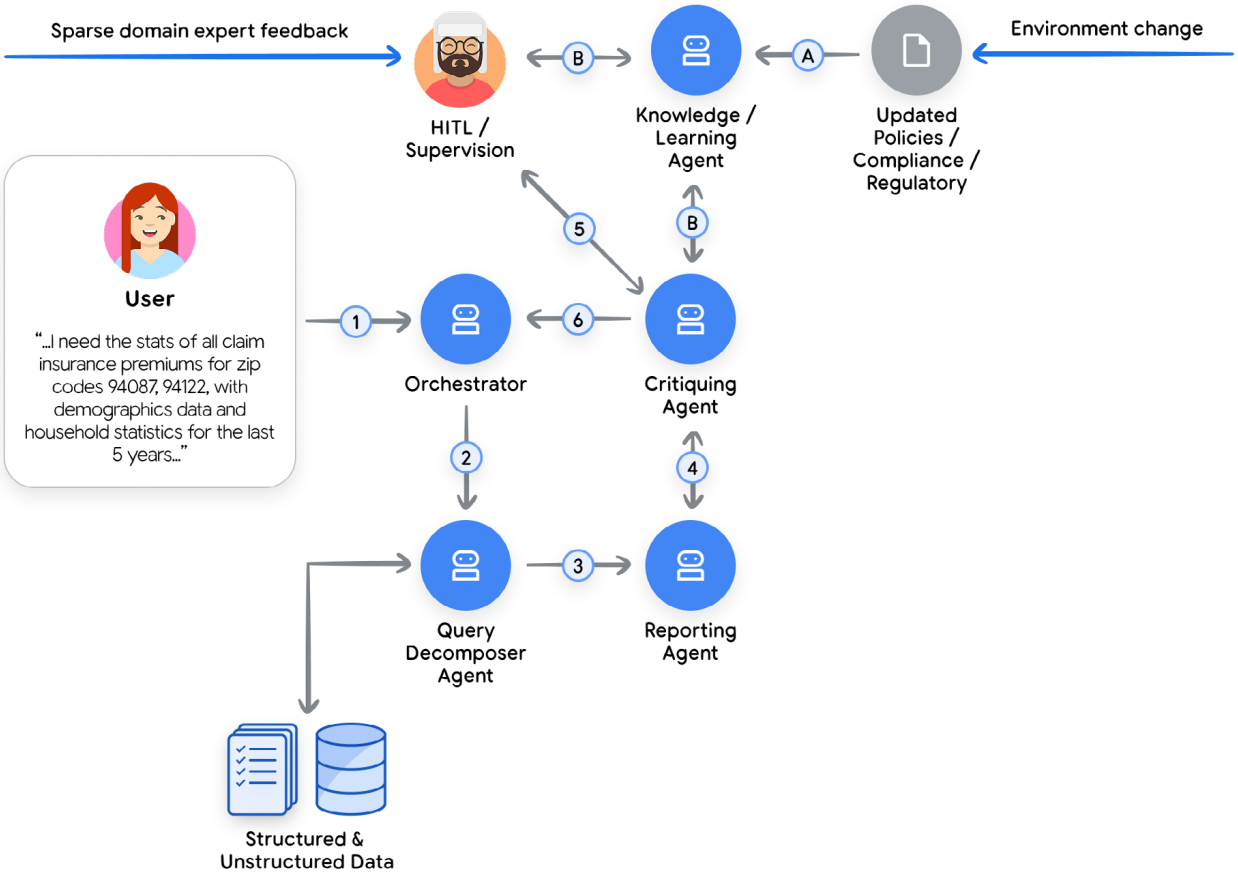
额外的优化技术，如动态重新配置多智能体设计模式或使用来自人类反馈的强化学习 (RLHF)，是活跃的研究领域。

示例：学习新的合规准则 (Example: Learning New Compliance Guidelines)

考虑一个在金融或生命科学等受到严格监管的行业中运营的企业智能体。其任务是生成必须符合隐私和监管规则（例如 GDPR）的报告。

这可以使用多智能体工作流来实现：

1. **查询智能体 (Querying Agent)** 响应用户请求检索原始数据。
2. **报告智能体 (Reporting Agent)** 将此数据综合成报告草稿。
3. **批评智能体 (Critiquing Agent)** 配备已知的合规准则，审查报告。如果它遇到歧义或需要最终签署，它会升级给人类领域专家。
4. **学习智能体 (Learning Agent)** 观察整个交互，特别注意来自人类专家的纠正性反馈。然后，它将此反馈概括为一个新的、可重用的准则（例如，批评智能体的更新规则或报告智能体的完善上下文）。



例如，如果人类专家指出某些家庭统计数据必须匿名化，学习智能体将记录此更正。下次生成类似报告时，批评智能体将自动应用此新规则，减少对人工干预的需求。这种批评、人类反馈和概括的循环允许系统自主适应不断变化的合规要求。

模拟与 Agent Gym ——下一个前沿 (Simulation and Agent Gym - the next frontier)

我们提出的设计模式可以归类为在线学习，即智能体需要利用它们被设计时的资源和设计模式进行学习。现在正在研究更先进的方法，其中有一个专用平台，旨在利用先进的工具和能力在离线过程中优化多智能体系统，这些工具和能力不是多智能体运行时环境的一部分。这种 Agent Gym 的关键属性是：

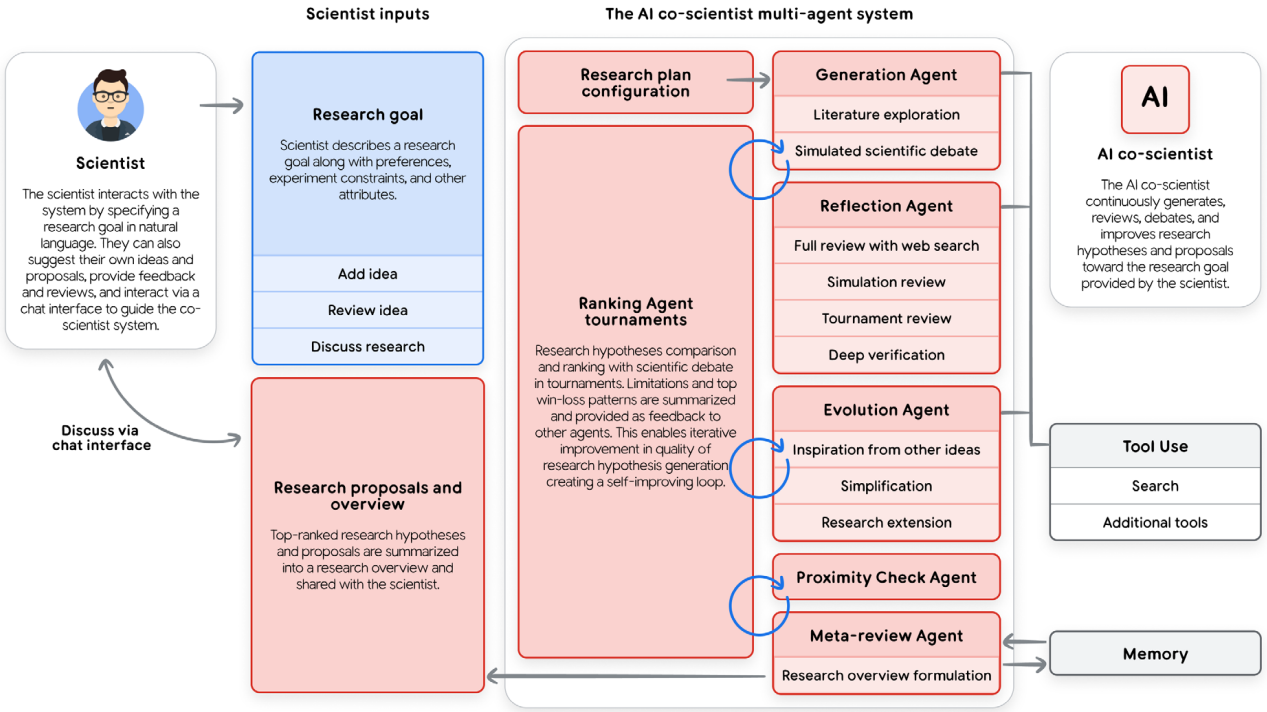
1. 它不在执行路径中。它是一个独立的非生产平台，因此可以拥有任何 LM 模型、离线工具、云应用程序等的协助。
2. 它提供了一个模拟环境，以便智能体可以在新数据上“锻炼”并学习。这个模拟环境非常适合具有许多优化路径的“试错”。
3. 它可以调用高级合成数据生成器，引导模拟尽可能真实，并对智能体进行压力测试（这可以包括高级技术，如红队测试 Red-teaming、动态评估和一系列批评智能体）。
4. 优化工具的武库不是固定的，它可以采用新工具（通过像 MCP 或 A2A 这样的开放协议），或者在更高级的设置中——学习新概念并围绕它们制作工具。
5. 最后，即使是像 Agent Gym 这样的构造，也可能无法克服某些边缘情况（由于企业中众所周知的“部落知识”问题）。在这些情况下，我们看到 Agent Gym 能够连接到领域专家的人类网络，并就正确的结果集咨询他们，以指导下一组优化。

高级智能体示例 (Examples of advanced agents)

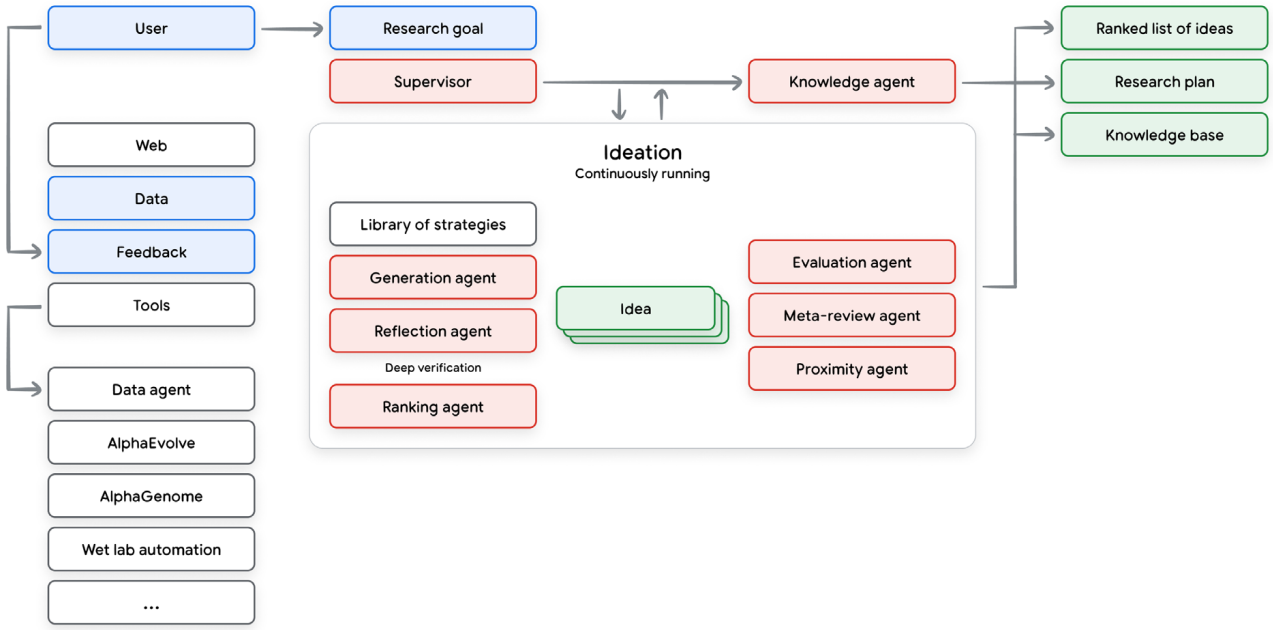
Google Co-Scientist

Co-Scientist 是一个高级 AI 智能体，旨在充当虚拟研究合作者，通过系统地探索复杂的问题空间来加速科学发现。它使研究人员能够定义目标，将智能体建立在指定的公共和专有知识源中，然后生成并评估一系列新颖的假设。

为了能够实现这一目标，Co-Scientist 衍生了一个完整的智能体生态系统，彼此协作。



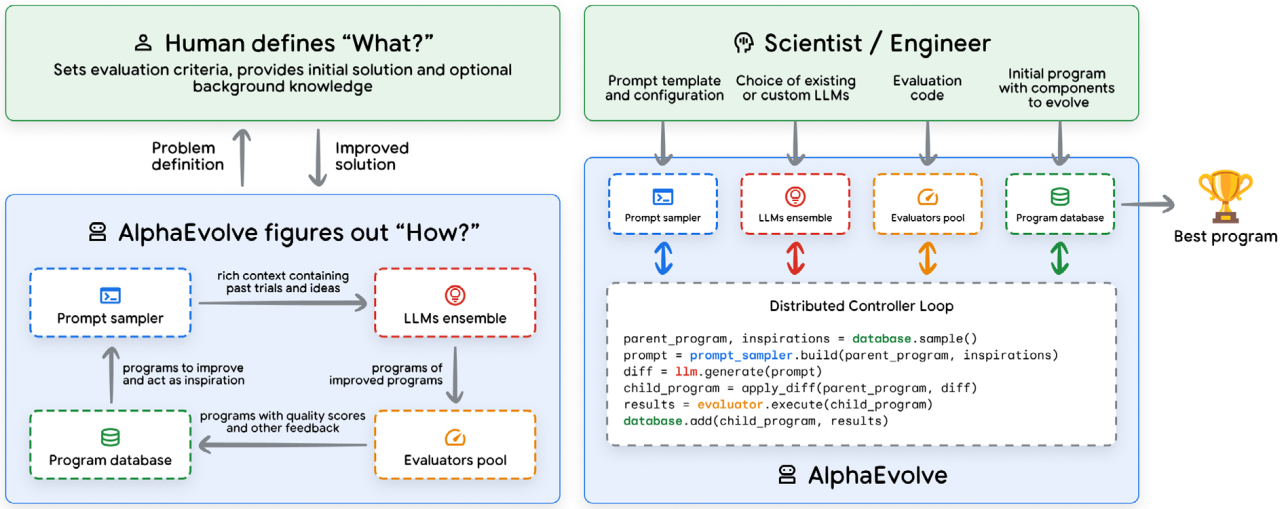
将系统视为一个研究项目经理。AI 首先采取广泛的研究目标并创建一个详细的项目计划。然后，“主管 (Supervisor)”智能体充当经理，将任务委托给一组专门的智能体，并分配像计算能力这样的资源。这种结构确保项目可以轻松扩展，并且团队的方法在朝着最终目标努力的过程中得到改进。



各种智能体工作数小时，甚至数天，并不断改进生成的假设，运行循环和元循环，不仅改进产生的想法，还改进我们判断和创造新想法的方式。

AlphaEvolve 智能体（AlphaEvolve Agent）

高级智能体系统的另一个例子是 AlphaEvolve，这是一个发现并优化数学和计算机科学中复杂问题算法的 AI 智能体。

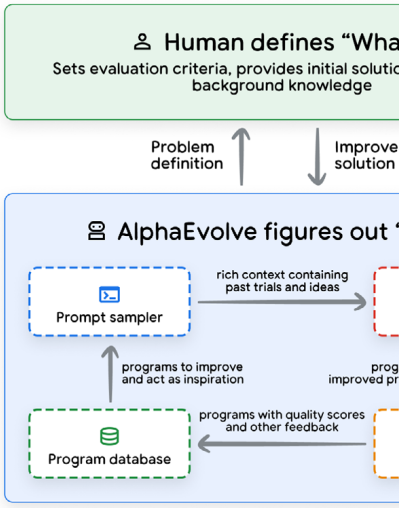


AlphaEvolve 结合了我们 Gemini 语言模型的创造性代码生成与自动化评估系统。它使用一个进化过程：AI 生成潜在的解决方案，评估器对它们进行评分，最有希望的想法被用作下一代代码的灵感。

这种方法已经导致了重大的突破，包括：

- 提高 Google 数据中心、芯片设计和 AI 训练的效率。

- 发现更快的矩阵乘法算法。
- 找到未解数学问题的新解决方案。



AlphaEvolve 擅长解决那些验证解决方案质量比从头开始找到解决方案容易得多的问题。

AlphaEvolve 专为人类与 AI 之间的深度、迭代伙伴关系而设计。这种合作主要通过两种方式进行：

- **透明的解决方案 (Transparent Solutions)**：AI 生成人类可读的代码作为解决方案。这种透明度允许用户理解逻辑，获得见解，信任结果，并直接修改代码以满足其需求。
- **专家指导 (Expert Guidance)**：人类专业知识对于定义问题至关重要。用户通过细化评估指标和引导探索来指导 AI，这防止了系统利用问题定义中的意外漏洞。这种交互循环确保最终解决方案既强大又实用。

智能体的结果是代码的持续改进，这种改进不断提高人类指定的指标。

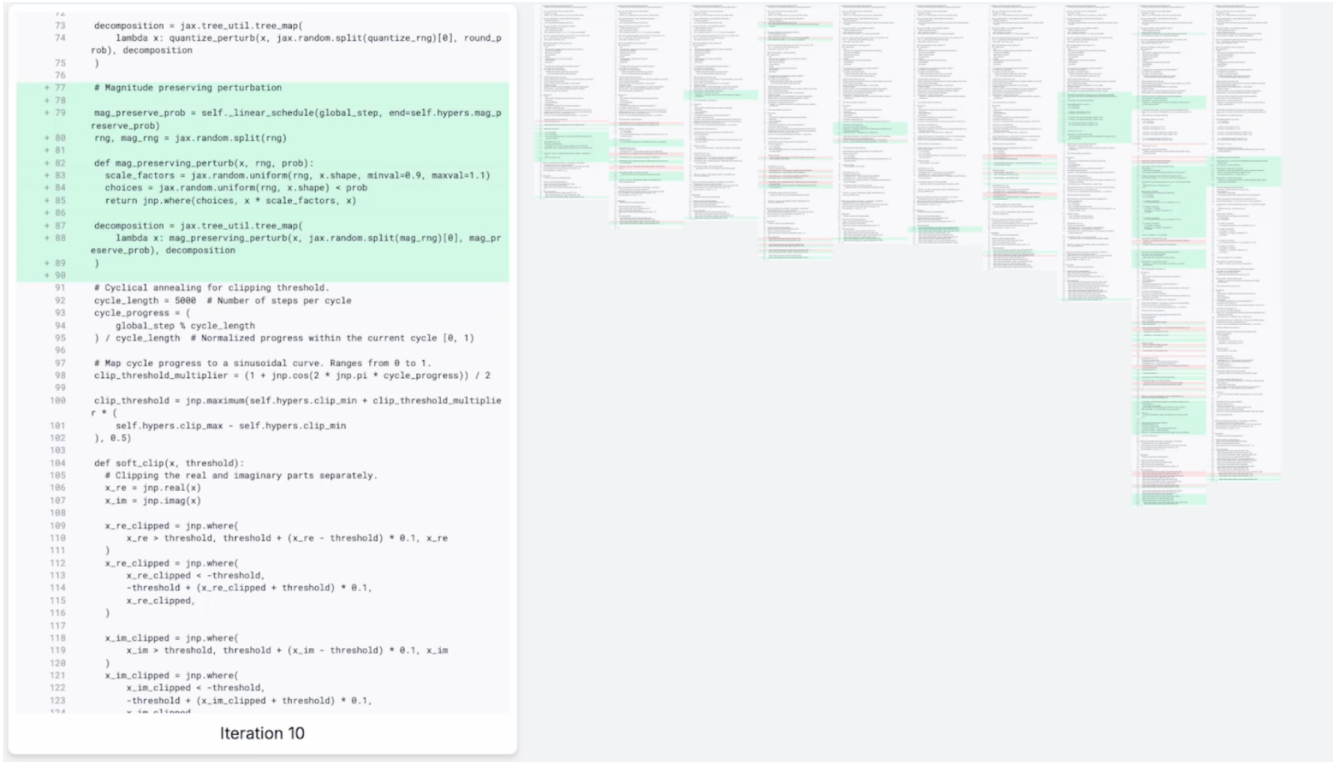


Figure 1: Figure 11: 算法演进

结论 (Conclusion)

生成式 AI 智能体标志着一个关键的演变，将人工智能从内容创作的被动工具转变为问题解决的主动、自主合作伙伴。本文档为理解和构建这些系统提供了一个正式框架，从原型过渡到建立可靠的、生产级的架构。

我们将智能体解构为其三个基本组件：推理模型（“大脑”）、可执行工具（“双手”）和管理编排层（“神经系统”）。正是这些部分的无缝集成，在连续的“思考、行动、观察”循环中运行，才释放了智能体的真正潜力。通过对智能体系统进行分类——从 Level 1 的联网问题解决者到 Level 3 的协作多智能体系统——架构师和产品负责人现在可以战略性地规划他们的雄心壮志，以匹配手头任务的复杂性。

核心挑战和机遇在于一个新的开发者范式。我们不再只是定义明确逻辑的“砖瓦匠”；我们是必须指导、约束和调试自主实体的“架构师”和“导演”。使 LM 如此强大的灵活性也是其不可靠性的来源。因此，成功不在于最初的提示本身，而在应用于整个系统的工程严谨性：在稳健的工具合同、弹性的错误处理、复杂的上下文管理和全面的评估中。

本文概述的原则和架构模式是基础蓝图，是我们在软件这一新领域中前行的路标，使我们能够构建的不仅是”工作流自动化”，而是真正具有协作精神、能力出众且适应性强的团队新成员。随着这项技术日臻成熟，这种严谨的架构方法将成为充分释放智能体 AI 潜能的决定性因素。

尾注 (Endnotes)

1. Julia Wiesinger, Patrick Marlow, et al. 2024 “Agents”. Available at: https://www.kaggle.com/whitepaper-agents.
2. Antonio Gulli, Lavi Nigam, et al. 2025 “Agents Companion”. Available at: https://www.kaggle.com/whitepaper-agent-companion.
3. Shunyu Yao, Y. et al., 2022, ‘ReAct: Synergizing Reasoning and Acting in Language Models’. Available at: https://arxiv.org/abs/2210.03629.
4. Wei, J., Wang, X. et al., 2023, ‘Chain-of-Thought Prompting Elicits Reasoning in Large Language Models’. Available at: https://arxiv.org/pdf/2201.11903.pdf.
5. Shunyu Yao, Y. et al., 2022, ‘ReAct: Synergizing Reasoning and Acting in Language Models’. Available at: https://arxiv.org/abs/2210.03629.
6. https://www.amazon.com/Agentic-Design-Patterns-Hands-Intelligent/dp/3032014018
7. Shunyu Yao, et. al., 2024, ‘ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains’, Available at: https://arxiv.org/abs/2406.12045.
8. https://artificialanalysis.ai/guide
9. https://cloud.google.com/vertex-ai/generative-ai/docs/model-reference/vertex-ai-model-optimizer
10. https://gemini.google/overview/gemini-live/
11. https://cloud.google.com/vision?e=48754805&hl=en
12. https://cloud.google.com/speech-to-text?e=48754805&hl=en
13. https://medium.com/google-cloud/genaiops-operationalize-generative-ai-a-practical-guide-d5bedaa59d78
14. https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/code-execution/overview
15. https://ai.google.dev/gemini-api/docs/function-calling
16. https://github.com/modelcontextprotocol/
17. https://ai.google.dev/gemini-api/docs/google-search
18. https://google.github.io/adk-docs/
19. https://google.github.io/adk-docs/sessions/memory/
20. https://cloud.google.com/architecture/choose-design-pattern-agentic-ai-system
21. https://cloud.google.com/vertex-ai/generative-ai/docs/agent-engine/overview
22. https://cloud.google.com/kubernetes-engine/docs/concepts/gke-and-cloud-run
23. https://github.com/GoogleCloudPlatform/agent-starter-pack
24. Sokratis Kartakis, 2024, ‘GenAI in Production: MLOps or GenAIOps?’. Available at: https://medium.com/ google-cloud/genai-in-production-mlops-or-genaiops-25691c9becd0.
25. Guangya Liu, Sujay Solomon, March 2025 “AI Agent Observability - Evolving Standards and Best Practice”. Available at: https://opentelemetry.io/blog/2025/ai-agent-observability/.
26. https://discuss.google.dev/t/agents-are-not-tools/192812
27. Damien Masson et. al, 2024, ‘DirectGPT: A Direct Manipulation Interface to Interact with Large Language Models’. Available at: https://arxiv.org/abs/2310.03691.
28. MCP UI is a system of controlling UI via MCP tools https://mcpui.dev/.
29. AG UI is a protocol of controlling UI via event passing and optionally shared state https://ag-ui.com/.
30. A2UI is a protocol of generating UI via structured output and A2A message passing https://github.com/google/A2UI.
31. https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash-live-api.
32. https://saif.google/focus-on-agents.
33. https://simonwillison.net/series/prompt-injection/.
34. https://storage.googleapis.com/gweb-research2023-media/pubtools/1018686.pdf.
35. https://spiffe.io/.
36. https://openreview.net/pdf?id=l9rATNBB8Y.
37. https://google.github.io/adk-docs/safety/. Introduction to Agents and Agent architectures

38. https://google.github.io/adk-docs/callbacks/design-patterns-and-best-practices /#guardrails-policy-enforcement
39. https://cloud.google.com/security-command-center/docs/model-armor-overview
40. https://cloud.google.com/vertex-ai/generative-ai/docs/provisioned-throughput/overview
41. https://cloud.google.com/run/sla
42. https://github.com/CharlesQ9/Self-Evolving-Agents
43. Juraj Gottweis, et. al., 2025, 'Accelerating scientific breakthroughs with an AI co-scientist' . Available at: https://research.google/blog/accelerating-scientific-breakthroughs-with-an-ai-co-scientist/.
44. Deepak Nathani et. al. 2025, 'MLGym: A New Framework and Benchmark for Advancing AI Research Agents' , Available at: https://arxiv.org/abs/2502.14499.

版权

Figure 2: 扫一扫关注”西滨 AI 随想”公众号和你一起学 AI