Claude 官方发布《构建高效的 Agents 指南》

在上一篇文章中，我们解读了 Claude 官方发布的研究报告

如何构建有效的 AIAgents：化繁为简——深度解读 Claude 实践《Building effective agents》(上)

很多朋友想通读 Anthropic 的这份原版报告，今天我为你带来这篇研究报告的中文翻译完整版。

导读部分

当我们畅想人工智能的未来时，很容易陷入一个直觉思维：

系统越复杂，功能越强大。

然而，Anthropic 最新发布的研究报告却给出了一个令人深思的答案 —— 在 AI 代理（AI Agents）的开发中，"Less is More"才是制胜法则。

过去一年，通过与数十个行业团队的密切合作，Anthropic 发现了一个有趣的现象：

最成功的 AI 代理系统并非建立在复杂的框架之上，而是采用简单、可组合的设计模式。

这个发现颠覆了许多开发者的固有认知。

想象一下，当你需要一个能自主完成复杂任务的 AI 助手时，你会怎么做？

是堆砌繁复的技术框架，还是从最基础的模块开始，逐步构建一个优雅精简的系统？

这个选择，可能比你想象的更重要。

为什么简单的设计反而能带来更好的效果？AI 代理的未来究竟在何方？

让我们跟随 Anthropic 的研究发现，一起探索 AI 开发中这个令人着迷的悖论……

在公众号后台发送口令“Agent”即可获取完整原版 PDF 报告。

接下来，我们来跟随 Anthropic 研究团队，从他们过去一年的实践中一起看看如何 “构建有效的 Agents”。

等等，在这之前，我们可能先得统一下对“Agents”的理解。这样才能更准确地理解 Anthropic 研究报告的意思。

最近 AI Agent 比较火，带动了很多人热衷讨论，就连续看到了一些比较奇葩的言论 ：

有人说他家的智能空调就是一个"数字孪生的 AI 智能体"。

看到这种说法，我忍不住想笑：照这么说，我家那台老洗衣机岂不是早就进化成"智能体"了？

其实，这反映出一个挺普遍的问题：**很多人对 Agents 这个概念理解得不够透彻。**

我们不妨从词源学的角度出发来理解"Agent"这个词。

它源自拉丁语"agere"，**意为"行动、执行、推动"，引申义为"代表他人做某事的人"。**

这个词在进入英语后，一直保持着"**代表某人或某个组织行动的人**"这层核心含义。

我们看看现实生活中的例子就更清楚了：

- FBI Agent（联邦调查局特工）——代表政府执行执法任务
- Travel Agent（旅行社代理人）——代表旅客安排行程
- Insurance Agent（保险代理人）——代表保险公司办理业务。

这些场景中的 Agent 都在扮演着"**代表他人做某事**"的角色。

正是基于这种"**代表他人做某事**"的本质含义，我们不应该把 AI Agent 生硬地翻译成"智能体"。

"智能体"这个词虽然听起来很高级，但完全丢失了"代理"这个核心概念。

更准确的译法应该是"**AI 代理人**"或"**AI 执行者**"

——它强调了这类 AI 系统的本质：**代表人类执行特定任务的数字化代理人。**

当我们真正理解了 Agents 的本质，对其应用也就有了更清晰的认识。

具体来说，**有三点关键洞察值得分享：**

第一点，它必须从实际工作流程出发。 任何一个靠谱的 AI Agent 项目，都应该先摸清楚现有流程中的痛点。比如，哪些环节特别耗费人力？哪些任务总是让团队叫苦不迭？这些才是 AI Agent 该着手解决的问题。

说完痛点，我们来**看价值，AI Agents 的核心价值在于实现从手动到自动的转变。**

它就像一个得力助手，接管那些原本需要人工重复操作的任务。

重点是"接管"而不是"创造" —— 我们要找的是已经存在且有价值的人工环节，而不是为了用 AI 而用 AI。

最后，**从商业角度来看**，也是最实在的一点：**投资回报率**。

把手动流程转化为自动化后，到底能省下多少人力成本？能提升多少效率？

这些都是**衡量 AI Agent 项目是否值得投入的关键指标。**

相比之下，那些本来就是自动化的系统，比如自动调温的空调，硬要贴上 AI Agents 的标签，不过是赶时髦罢了。

真正的 AI Agents 应该是帮你解决实际问题的数字助手，而不是概念上的哗众取宠。

这也正印证了 Anthropic 团队所强调的"Less is More"理念 —— **真正有价值的 AI Agents 往往始于对具体问题的深入理解，而不是华丽的技术包装。**

构建高效的 Agents

过去一年里，我们与来自不同行业的数十个团队合作，共同开发基于大语言模型（LLM）的代理系统。

我们发现一个持续的现象：最成功的实现并非依赖复杂的框架或专门的代码库，而是采用简单、可组合的模式构建。

在这篇文章中，我们将分享与客户合作以及自主构建代理系统过程中的经验，并为开发者提供一些实用建议。

什么是 Agents 代理?

"代理"（Agent）可以有多种定义方式。有些客户将代理定义为完全自主的系统，它们能够长期独立运行，利用各种工具完成复杂任务。

也有客户用这个术语来描述更具规范性的实现，即遵循预定义工作流程的系统。

在 Anthropic，我们将这些变体都归类为代理型系统，但在架构上区分了工作流和代理：

- **• 工作流是通过预定义的代码路径来编排 LLM 和工具的系统。**
- **• 而代理则是 LLM 能够动态指导自身的处理过程和工具使用的系统，它们能够自主控制任务的完成方式。**

接下来，我们将详细探讨这两类代理型系统。

在附录 1（"代理的实践应用"）中，我们会介绍客户在使用这些系统时发现特别有价值的两个领域。

何时（以及何时不）使用代理？

在构建基于 LLM 的应用时，我们建议**先寻找最简单的解决方案，只在必要时才增加复杂性。**这可能意味着完全不需要构建代理型系统。

代理型系统通常会用延迟和成本来换取更好的任务表现，您需要考虑这种取舍是否值得。**当确实需要更复杂的系统时，对于明确定义的任务，工作流能提供更好的可预测性和一致性**；**而当需要大规模的灵活性和模型驱动的决策时，代理则是更好的选择。**

不过，对于许多应用来说，优化单个 LLM 调用（配合检索和上下文示例）通常就足够了。

框架的使用时机和方式

目前有许多框架可以让代理系统的实现变得更简单，包括：

- 来自 LangChain 的 LangGraph；
- 亚马逊 Bedrock 的 AI 代理框架；
- Rivet，一个拖放式的 LLM 工作流构建工具；
- Vellum，另一个用于构建和测试复杂工作流的图形界面工具。

这些框架通过简化标准的底层任务（如调用 LLM、定义和解析工具、链接调用等），使入门变得容易。

但是，**它们往往会创建额外的抽象层，这可能会掩盖底层的提示和响应，使调试变得更困难。**它们也可能诱使开发者在一个更简单的设置就足够的情况下增加不必要的复杂性。

我们**建议开发者直接从使用 LLM API 开始：许多模式只需要几行代码就能实现。**

如果您确实要使用框架，请确保理解底层代码。对框架内部机制的错误假设是客户常见的错误来源。

具体实现示例请参考我们的操作指南。

构建模块、工作流和代理

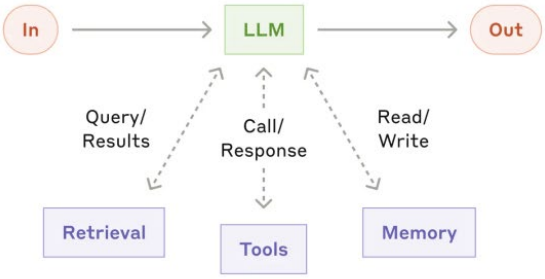
在这一节中，我们将探讨在生产环境中常见的代理系统模式。

我们将从基础构建模块 —— 增强型 LLM 开始，逐步增加复杂度，从简单的组合工作流到自主代理。

构建模块：增强型 LLM

代理系统的基本构建模块是经过增强的 LLM，它具备检索、工具使用和记忆等功能。

我们当前的模型可以主动使用这些能力 —— 生成自己的搜索查询、选择合适的工具，并决定保留哪些信息。



The augmented LLM

对于实现，我们建议重点关注两个方面：

将这些功能针对性地适配您的具体用例，并确保为 LLM 提供简单、文档完备的接口。

虽然实现这些增强功能的方式有很多，

但其中一种方法是通过我们最近发布的模型上下文协议（Model Context Protocol），

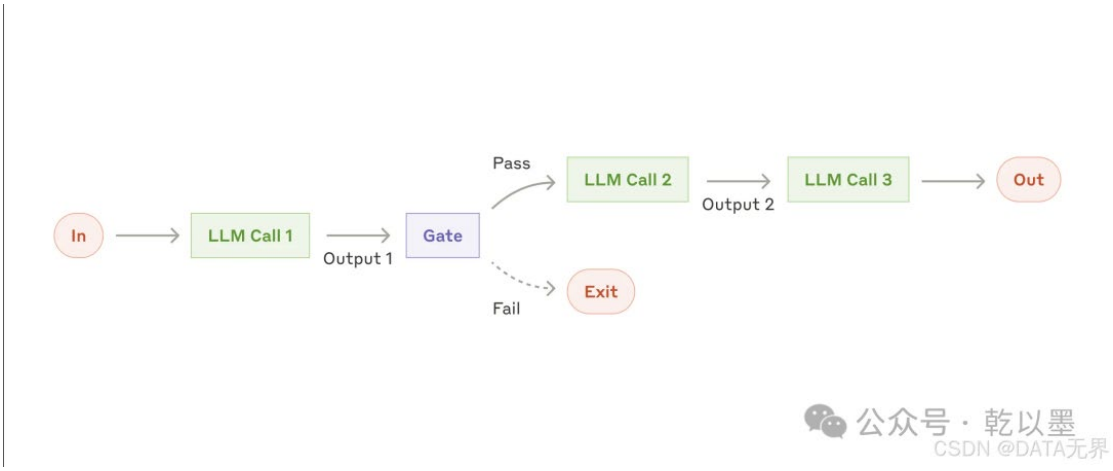
它允许开发者通过简单的客户端实现来集成不断增长的第三方工具生态系统。

在本文的剩余部分，我们将假设每个 LLM 调用都能访问这些增强功能。

工作流：提示链

提示链将任务分解为一系列步骤，每个 LLM 调用都会处理前一个调用的输出。

您可以在任何中间步骤添加程序化检查（参见下图中的"gate"），以确保整个流程仍在正轨上。



The prompt chaining workflow

工作流的使用时机：这种工作流最适合那些可以轻松、清晰地分解为固定子任务的情况。

主要目标是通过让每个 LLM 调用处理更简单的任务来用延迟换取更高的准确性。

提示链的实用示例：

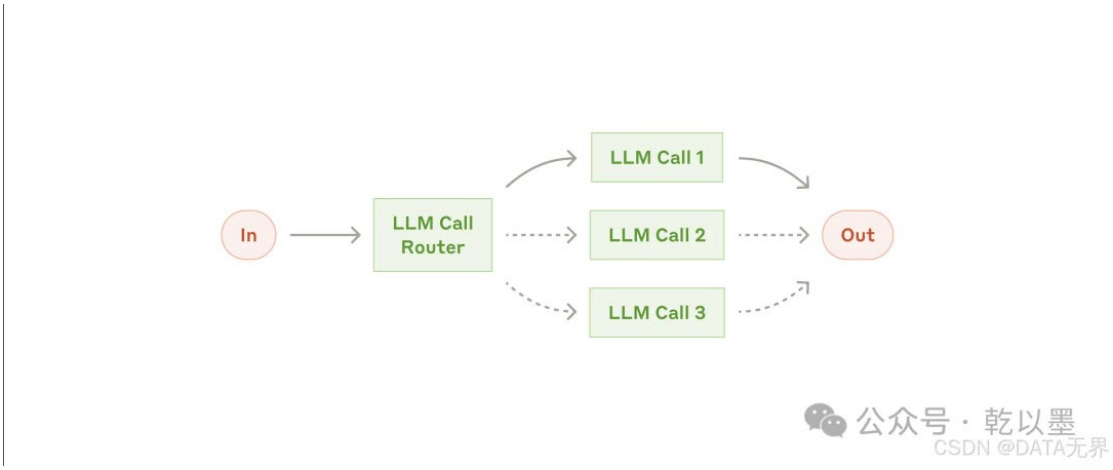
- 生成营销文案，然后将其翻译成其他语言
- 撰写文档大纲，检查大纲是否符合特定标准，然后基于大纲撰写文档

工作流：路由

路由会对输入进行分类，并将其引导至专门的后续任务。

这种工作流允许关注点分离，并构建更专门化的提示。

如果没有这种工作流，为某一类型输入优化可能会影响到其他输入的性能表现。



The routing workflow

使用时机：路由适用于那些有明显不同类别、需要分开处理的复杂任务，且这些类别可以通过 LLM 或传统的分类模型/算法进行准确分类。

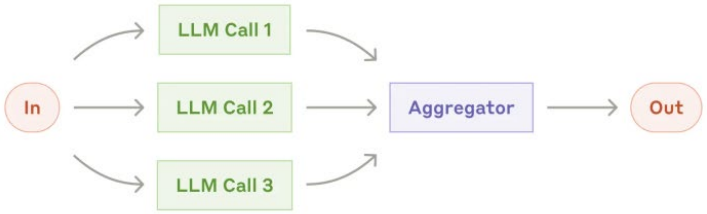
路由的实用示例：

- • 将不同类型的客服查询（一般问题、退款请求、技术支持）引导到不同的下游流程、提示和工具
- • 将简单/常见问题路由给较小的模型（如 Claude 3.5 Haiku），将困难/特殊问题路由给更强大的模型（如 Claude 3.5 Sonnet），以优化成本和速度

工作流：并行化

LLM 有时可以同时处理任务，并通过程序化方式汇总其输出。这种并行化工作流主要有两种关键变体：

- **• 分段：将任务拆分为并行运行的独立子任务**
- **• 投票：多次运行相同任务以获得多样化的输出**



The parallelization workflow

使用时机:

- **• 并行化在以下情况下特别有效：**
  - **• 一是可以将子任务并行化以提高速度**
  - **• 二是需要多个视角或尝试来获得更高置信度的结果。**
- **• 对于涉及多个考虑因素的复杂任务，当每个考虑因素由单独的 LLM 调用处理时，LLM 通常表现更好，因为这允许对每个具体方面进行专注处理。**

并行化的实用示例：

分段应用：

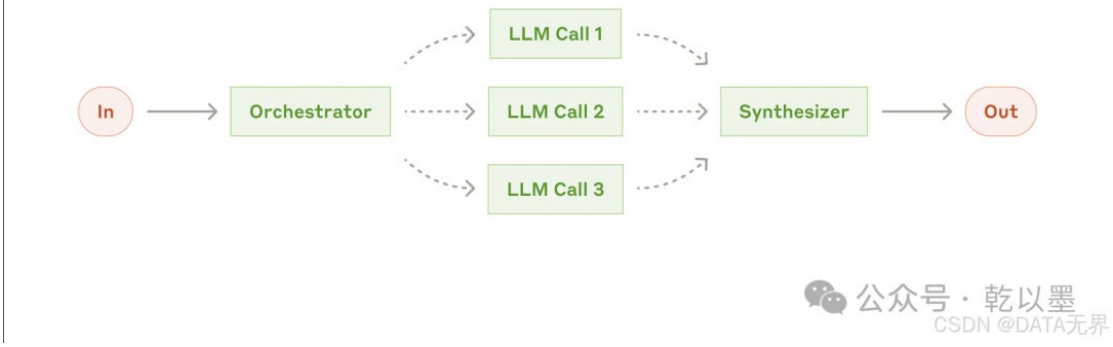
- **• 实现安全防护，**其中一个模型实例处理用户查询，而另一个筛查不当内容或请求。这比让同一个 LLM 调用同时处理安全防护和核心响应的效果更好
- **• 自动化评估 LLM 性能，**每个 LLM 调用评估模型在给定提示下的不同表现方面。

投票应用：

- **• 审查代码中的漏洞，**多个不同的提示审查代码并在发现问题时进行标记。
- **• 评估给定内容是否不当，**多个提示评估不同方面，或要求不同的投票阈值来平衡误报和漏报。

工作流：编排者-执行者

在编排者-执行者工作流中，一个中央 LLM 动态地分解任务，将它们分配给执行者 LLM，并综合它们的结果。



The orchestrator-workers workflow

使用时机:

这种**工作流非常适合那些无法预测所需子任务的复杂任务（例如，在编程中，需要修改的文件数量和每个文件中的修改性质可能都取决于具体任务）。**

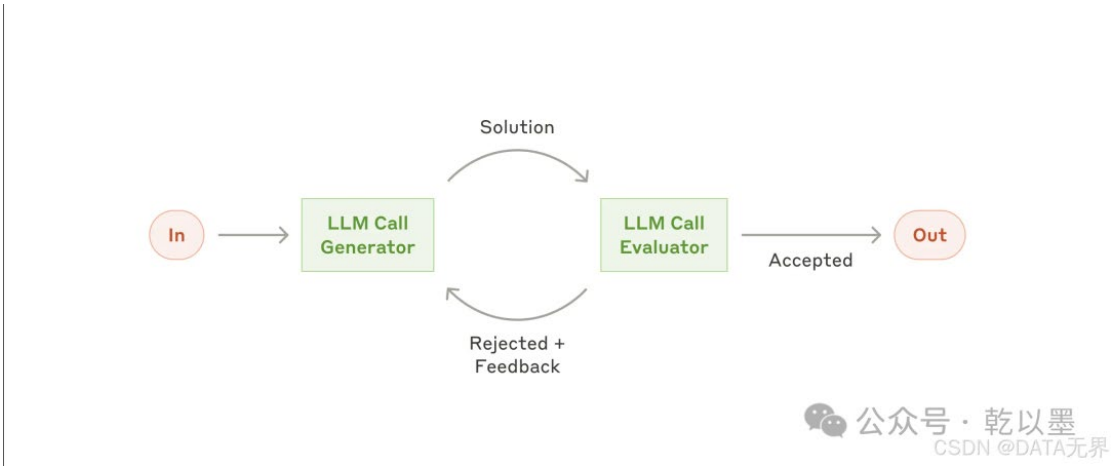
虽然在结构上与并行化类似，但关键区别在于其灵活性 —— 子任务不是预先定义的，而是由编排者根据具体输入来确定。

编排者-执行者工作流的实用示例：

- 需要对多个文件进行复杂修改的编程产品
- 涉及从多个来源收集和分析可能相关信息的搜索任务

工作流：评估者-优化者

在评估者-优化者工作流中，一个 LLM 调用生成响应，而另一个在循环中提供评估和反馈。



The evaluator-optimizer workflow

使用时机:

当我们有明确的评估标准，且迭代优化能带来可衡量的价值时，这种工作流特别有效。

判断是否适合使用这种工作流有两个标志:

- **• 首先，当人类明确表达反馈时，LLM 的响应能够明显改善；**
- **• 其次，LLM 能够提供这样的反馈。这类似于人类作者在创作精良文档时可能经历的迭代写作过程。**

评估者-优化者工作流的实用示例：

- **• 文学翻译，**其中译者 LLM 最初可能无法捕捉到的细微差别可以通过评估者 LLM 的有用批评来改进。
- **• 需要多轮搜索和分析才能收集全面信息的复杂搜索任务**，其中评估者决定是否需要进行更多搜索。

代理

随着 LLM 在关键能力上日趋成熟 —— 理解复杂输入、进行推理和规划、可靠地使用工具以及从错误中恢复，代理系统正在生产环境中逐渐兴起。

代理通过接收用户的指令或与用户进行交互对话来开始工作。

一旦任务明确，代理就会独立规划和运行，必要时会向用户寻求更多信息或判断。

在执行过程中，代理需要在每个步骤都从环境中获取"基准事实"（如工具调用结果或代码执行情况）来评估其进展。

代理可以在检查点或遇到障碍时暂停等待人类反馈。

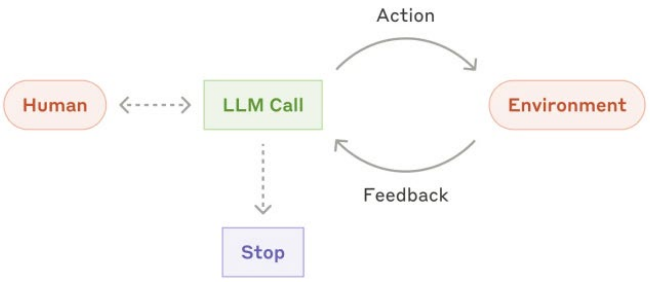
任务通常在完成时终止，但也常常包含停止条件（如最大迭代次数）以保持控制。

代理可以处理复杂的任务，但其实现往往很直接。

它们**通常就是基于环境反馈在循环中使用工具的 LLM。**

因此，**清晰、周到地设计工具集及其文档至关重要。**

我们在附录 2（"工具的提示工程"）中详细展开了工具开发的最佳实践。



Autonomous agent

使用时机:

代理适用于难以或无法预测所需步骤数量的开放性问题，以及无法硬编码固定路径的场景。LLM 可能需要运行多个回合，因此您必须对其决策能力有一定程度的信任。代理的自主性使其非常适合在可信环境中扩展任务。

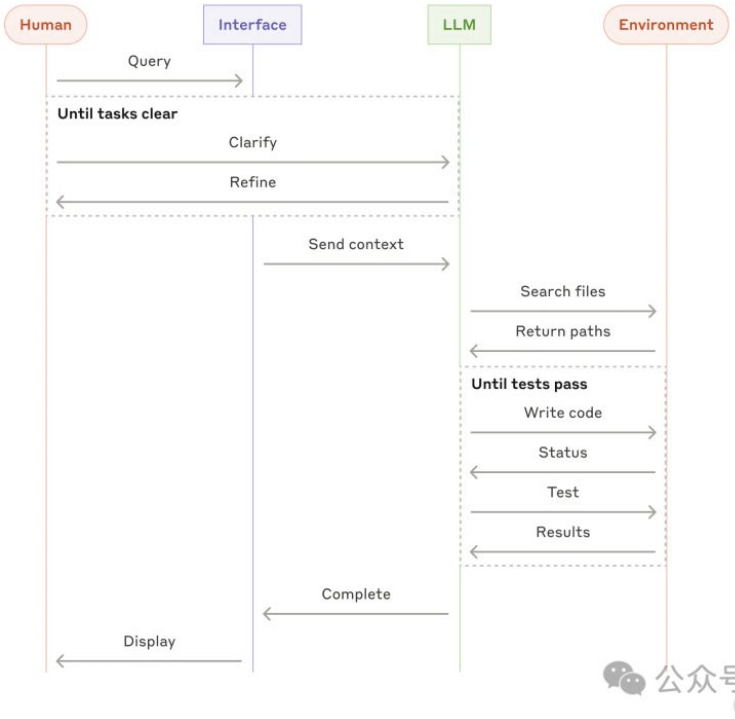
代理的自主特性意味着更高的成本，以及错误累积的可能性。

我们建议在沙盒环境中进行广泛测试，并配备适当的安全防护措施。

代理的实用示例：

以下是我们自己实现的例子：

- • 一个用于解决 SWE-bench 任务的编码代理，该任务涉及根据任务描述对多个文件进行编辑
- • " "



High-level flow of a coding agent

组合和定制这些模式

这些构建模块并非强制性的规范。它们是开发者可以根据不同用例来塑造和组合的常见模式。

与任何 LLM 功能一样，**成功的关键在于衡量性能并迭代实现。**

需要重申的是：**只有在确实能够改善结果的情况下，才考虑增加复杂性。**

总结

在 LLM 领域取得成功并不在于构建最复杂的系统，而在于构建最适合您需求的系统。

从简单的提示开始，通过全面的评估来优化它们，只有在更简单的解决方案不足以满足需求时，才添加多步骤的代理系统。

在实现代理时，我们遵循**三个核心原则：**

- **• 保持代理设计的简单性**
- **• 通过明确展示代理的规划步骤来确保透明度**
- **• 通过全面的工具文档和测试来精心设计代理-计算机接口（ACI）**

框架可以帮助您快速入门，但当转向生产环境时，不要犹豫于减少抽象层并使用基本组件构建。

通过遵循这些原则，您可以创建既强大又可靠、可维护，并能获得用户信任的代理系统。

致谢

本文由 Erik Schluntz 和 Barry Zhang 撰写。

这项工作源于我们在 Anthropic 构建代理的经验，以及客户分享的宝贵见解，我们对此深表感谢。

附录 1：代理的实践应用

通过与客户合作，我们发现了两个特别有前景的 AI 代理应用领域，它们展示了上述模式的实际价值。

这两个应用都说明了代理在以下场景中最能体现价值：

需要对话和行动相结合、有明确的成功标准、能够实现反馈循环，并整合有意义的人类监督。

A. 客户支持

客户支持将熟悉的聊天机器人界面与通过工具集成实现的增强功能相结合。

这非常适合更开放式的代理，原因如下：

- 支持交互自然地遵循对话流程，同时需要访问外部信息和执行操作
- 可以集成工具来获取客户数据、订单历史和知识库文章
- 发放退款或更新工单等操作可以通过程序化方式处理
- 可以通过用户定义的解决方案来清晰地衡量成功

几家公司已经通过基于使用量的定价模型证明了这种方法的可行性，他们只对成功解决的案例收费，这显示了他们对代理效能的信心。

B. 编码代理

软件开发领域展现出了 LLM 功能的巨大潜力，其能力已从代码补全发展到自主问题解决。

代理在这里特别有效，原因如下：

- 代码解决方案可以通过自动化测试进行验证
- 代理可以使用测试结果作为反馈来迭代解决方案
- 问题空间定义明确且结构化
- 输出质量可以客观衡量

在我们自己的实现中，代理现在可以仅基于拉取请求描述来解决 SWE-bench 验证基准中的真实 GitHub 问题。

然而，虽然自动化测试有助于验证功能，但人工审查对于确保解决方案符合更广泛的系统要求仍然至关重要。

附录 2：工具的提示工程

无论您构建的是哪种代理系统，工具都可能是代理的重要组成部分。

工具能够通过在我们的 API 中指定其确切的结构和定义，使 Claude 能够与外部服务和 API 交互。

当 Claude 响应时，如果它计划调用工具，API 响应中将包含一个工具使用块。

工具的定义和规范应该得到与整体提示同等的提示工程关注。

在这个简短的附录中，我们将描述如何对工具进行提示工程。

通常有几种方式可以指定相同的操作。

例如，您可以通过编写差异（diff）或重写整个文件来指定文件编辑。

对于结构化输出，您可以在 markdown 或 JSON 中返回代码。

在软件工程中，这些差异都是表面的，可以无损地相互转换。

然而，某些格式对 LLM 来说比其他格式更难写出。

编写差异需要在写入新代码之前知道块头中要更改的行数。

相比于在 markdown 中写代码，在 JSON 中写代码需要额外转义换行符和引号。

关于决定工具格式，我们的建议如下：

- • 给模型足够的词元（token）来"思考"，避免把自己写进死角
- • 保持格式接近模型在互联网上自然遇到的文本
- • 确保没有格式"开销"，比如必须保持对数千行代码的准确计数，或对任何代码进行字符串转义

一个经验法则是，考虑到人机界面（HCI）需要投入多少努力，

就计划在创建良好的代理-计算机接口（ACI）上投入同样多的努力。

以下是一些实现方法：

- **• 站在模型的角度思考。** 基于描述和参数，使用这个工具是否显而易见，还是需要仔细思考？如果是后者，那么对模型来说可能也是如此。一个好的工具定义通常包括使用示例、边界情况、输入格式要求，以及与其他工具的明确界限。
- **• 如何改变参数名称或描述使其更明显？** 可以把这想象成为团队中的初级开发者写