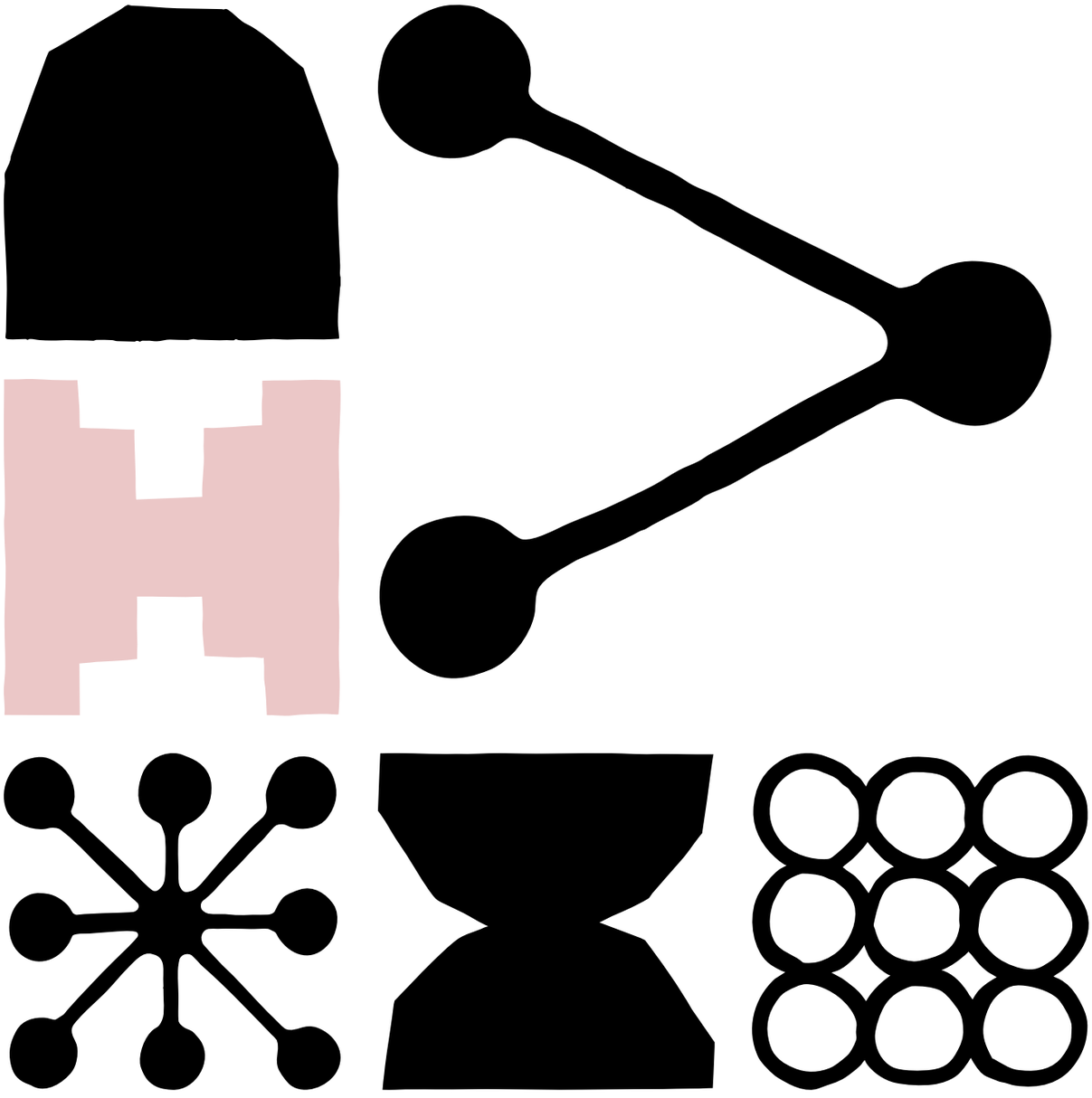
Engineering at Anthropic Anthropic



Writing effective tools for agents — with agents

Agents are only as effective as the tools we give them. We share how to write high-quality tools and evaluations, and how you can boost performance by using Claude to optimize its tools for itself.

Claude

Published Sep 11, 2025 2025 9 11

The Model Context Protocol (MCP) can empower LLM agents with potentially hundreds of tools to solve real-world tasks. But how do we make those tools maximally effective?

模型上下文协议（MCP）能够为大型语言模型智能体提供可能多达数百种的工具，以解决现实世界中的任务。但我们如何才能让这些工具发挥最大效用呢？

In this post, we describe our most effective techniques for improving performance in a variety of agentic AI systems[1].

在这篇文章中，我们介绍了在各种智能体人工智能系统中提升性能的最有效技术[1]。

We begin by covering how you can:

我们首先会介绍如何：

- Build and test prototypes of your tools

  构建并测试你的工具原型
- Create and run comprehensive evaluations of your tools with agents

  通过智能体创建并运行对工具的全面评估
- Collaborate with agents like Claude Code to automatically increase the performance of your tools

  与Claude Code等智能体合作，自动提升你的工具性能

We conclude with key principles for writing high-quality tools we've identified along the way:

最后，我们总结了在这一过程中确定的编写高质量工具的关键原则：

- Choosing the right tools to implement (and not to implement)

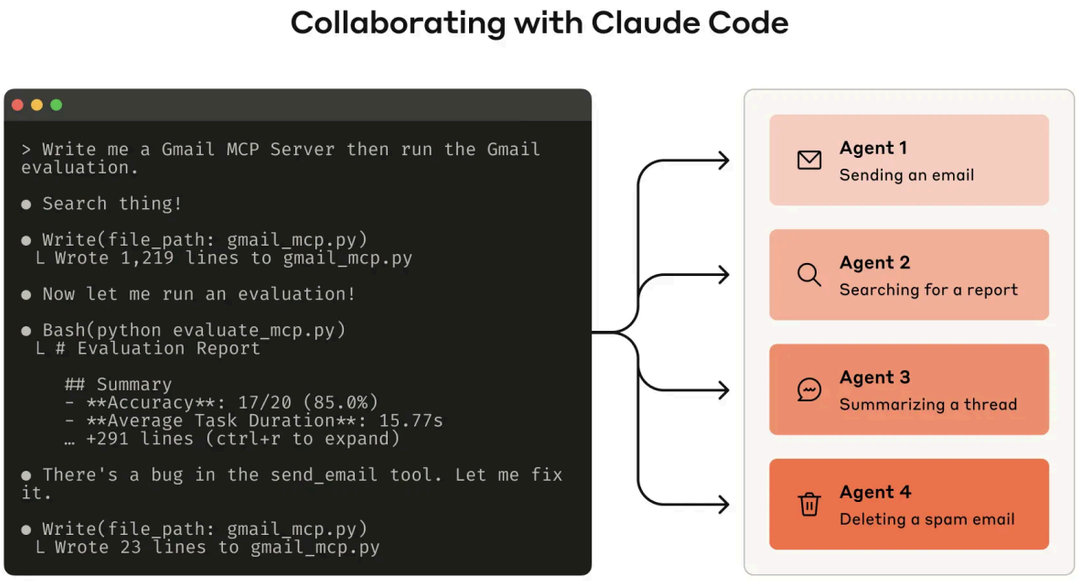
  选择合适的工具来实施（以及不实施哪些工具）
- Namespacing tools to define clear boundaries in functionality

  使用命名空间工具来明确功能边界
- Returning meaningful context from tools back to agents

  将工具中有意义的上下文返回给智能体
- Optimizing tool responses for token efficiency

  优化工具响应以提高令牌效率
- Prompt-engineering tool descriptions and specs

  提示词工程工具的描述和规格



Building an evaluation allows you to systematically measure the performance of your tools. You can use Claude Code to automatically optimize your tools against this evaluation.

Claude Code

What is a tool?

In computing, deterministic systems produce the same output every time given identical inputs, while *non-deterministic* systems—like agents—can generate varied responses even with the same starting conditions.

在计算领域，确定性系统在给定相同输入时每次都会产生相同的输出，而*非确定性*系统（如智能体）即使在初始条件相同的情况下，也可能生成不同的响应。

When we traditionally write software, we're establishing a contract between deterministic systems. For instance, a function call like `getWeather("NYC")` will always fetch the weather in New York City in the exact same manner every time it is called.

在传统的软件开发中，我们是在确定性系统之间建立一种约定。例如，像 `getWeather("NYC")` 这样的函数调用，每次被调用时，都会以完全相同的方式获取纽约市的天气。

Tools are a new kind of software which reflects a contract between deterministic systems and non-deterministic agents. When a user asks "Should I bring an umbrella today?," an agent might call the weather tool, answer from general knowledge, or even ask a clarifying question about location first. Occasionally, an agent might hallucinate or even fail to grasp how to use a tool.

工具是一种新型软件，它反映了确定性系统与非确定性智能体之间的约定。当用户问"我今天应该带伞吗？"时，智能体可能会调用天气工具、凭借常识回答，甚至先询问关于地点的澄清问题。有时，智能体可能会产生幻觉，甚至无法理解如何使用工具。

This means fundamentally rethinking our approach when writing software for agents: instead of writing tools and MCP servers the way we'd write functions and

APIs for other developers or systems, we need to design them for agents.

这意味着在为智能体编写软件时，我们需要从根本上重新思考我们的方法：我们不应像为其他开发者或系统编写函数和API那样编写工具和MCP服务器，而需要为智能体设计它们。

Our goal is to increase the surface area over which agents can be effective in solving a wide range of tasks by using tools to pursue a variety of successful strategies. Fortunately, in our experience, the tools that are most “ergonomic” for agents also end up being surprisingly intuitive to grasp as humans.

我们的目标是通过使用工具来采用各种成功策略，从而扩大智能体能够有效解决各类任务的范围。幸运的是，根据我们的经验，那些对智能体而言最“易于操作”的工具，最终也会出人意料地让人类觉得易于理解。

How to write tools

In this section, we describe how you can collaborate with agents both to write and to improve the tools you give them. Start by standing up a quick prototype of your tools and testing them locally. Next, run a comprehensive evaluation to measure subsequent changes. Working alongside agents, you can repeat the process of evaluating and improving your tools until your agents achieve strong performance on real-world tasks.

在本节中，我们将介绍如何与智能体协作，共同编写并改进为它们提供的工具。首先，快速搭建工具的简易原型并在本地进行测试。接下来，进行全面评估以衡量后续的变更。通过与智能体协作，你可以重复评估和改进工具的过程，直到智能体在实际任务中表现出色。

Building a prototype

It can be difficult to anticipate which tools agents will find ergonomic and which tools they won’t without getting hands-on yourself. Start by standing up a quick prototype of your tools. If you’re using Claude Code to write your tools (potentially in one-shot), it helps to give Claude documentation for any software libraries, APIs, or SDKs (including potentially the MCP SDK) your tools will rely on. LLM-friendly documentation can commonly be found in flat `llms.txt` files on official documentation sites (here’s our API’s).

如果不亲身体验，很难预测智能体会觉得哪些工具好用，哪些不好用。首先快速搭建你的工具原型。如果你使用Claude Code（可能通过一次性生成的方式）来编写工具，向Claude提供你的工具将依赖的任何软件库、API或SDK（可能包括MCP SDK）的文档会很有帮助。在官方文档网站的 `llms.txt` 扁平文件中，通常可以找到对大语言模型友好的文档（这是我们的API的文档）。

Wrapping your tools in a local MCP server or Desktop extension (DXT) will allow you to connect and test your tools in Claude Code or the Claude Desktop app.

将你的工具封装在本地MCP服务器或桌面扩展程序（DXT）中，你就可以在Claude Code或Claude桌面应用中连接并测试这些工具。

To connect your local MCP server to Claude Code, run `claude mcp add <name> <command> [args...]`.

要将您的本地MCP服务器连接到Claude Code，请运行 `claude mcp add <name> <command> [args...]` 。

To connect your local MCP server or DXT to the Claude Desktop app, navigate to `Settings > Developer` or `Settings > Extensions`, respectively.

要将本地MCP服务器或DXT连接到Claude桌面应用，请分别导航至 `Settings > Developer` 或 `Settings > Extensions` 。

Tools can also be passed directly into Anthropic API calls for programmatic testing.

工具也可以直接传入Anthropic API调用中，用于程序化测试。

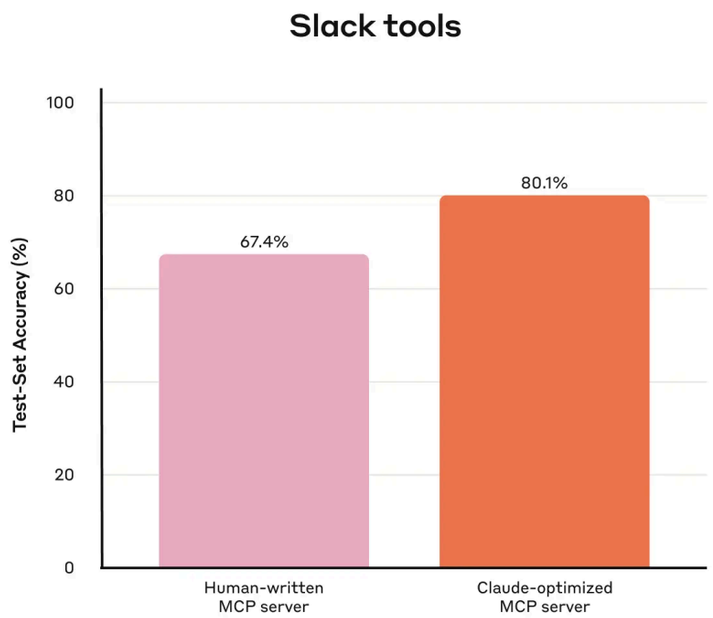
Test the tools yourself to identify any rough edges. Collect feedback from your users to build an intuition around the use-cases and prompts you expect your tools to enable.

自己测试这些工具，找出任何不完善之处。收集用户的反馈，以形成对工具所能支持的使用场景和提示词的直觉认知。

Running an evaluation

Next, you need to measure how well Claude uses your tools by running an evaluation. Start by generating lots of evaluation tasks, grounded in real world uses. We recommend collaborating with an agent to help analyze your results and determine how to improve your tools. See this process end-to-end in our tool evaluation cookbook.

接下来，你需要通过运行评估来衡量Claude使用你的工具的效果。首先，要生成大量基于现实应用的评估任务。我们建议与一个智能体合作，以帮助分析结果并确定如何改进你的工具。在我们的工具评估指南中可以看到这个完整的流程。



Held-out test set performance of our internal Slack tools

Slack

Generating evaluation tasks 生成评估任务

With your early prototype, Claude Code can quickly explore your tools and create dozens of prompt and response pairs. Prompts should be inspired by real-world uses and be based on realistic data sources and services (for example, internal knowledge bases and microservices). We recommend you avoid overly simplistic or superficial “sandbox” environments that don’t stress-test your tools with sufficient complexity. Strong evaluation tasks might require multiple tool calls—potentially dozens.

借助您的早期原型，Claude Code 可以快速探索您的工具，并创建数十个提示词和响应对。提示词应从实际应用中获取灵感，并基于真实的数据源和服务（例如，内部知识库和微服务）。我们建议您避免过于简单或肤浅的“沙盒”环境，这类环境无法以足够的复杂度对您的工具进行压力测试。出色的评估任务可能需要多次工具调用——甚至可能多达数十次。

Here are some examples of strong tasks:

以下是一些优秀任务的示例：

- Schedule a meeting with Jane next week to discuss our latest Acme Corp project. Attach the notes from our last project planning meeting and reserve a conference room.

  下周和简安排一次会议，讨论我们最新的Acme Corp项目。附上我们上次项目规划会议的笔记，并预订一间会议室。

- Customer ID 9182 reported that they were charged three times for a single purchase attempt. Find all relevant log entries and determine if any other customers were affected by the same issue.

  客户ID 9182报告称，他们在一次购买尝试中被收费三次。请找出所有相关的日志条目，并确定是否有其他客户受到同样问题的影响。

- Customer Sarah Chen just submitted a cancellation request. Prepare a retention offer. Determine: (1) why they're leaving, (2) what retention offer would be most compelling, and (3) any risk factors we should be aware of before making an offer.

  客户陈莎拉刚刚提交了取消请求。请准备一份挽留方案。需确定：（1）他们为何要离开；（2）哪种挽留方案最有吸引力；（3）在提出方案前我们应注意的任何风险因素。

And here are some weaker tasks: 以下是一些较弱的任务：

- Schedule a meeting with jane@acme.corp next week.

  下周与jane@acme.corp安排一次会议。

- Search the payment logs for `purchase_complete` and `customer_id=9182`.

  在支付日志中搜索 `purchase_complete` 和 `customer_id=9182` 。

- Find the cancellation request by Customer ID 45892.

  通过客户ID 45892查找取消请求。

Each evaluation prompt should be paired with a verifiable response or outcome. Your verifier can be as simple as an exact string comparison between ground truth and sampled responses, or as advanced as enlisting Claude to judge the response. Avoid overly strict verifiers that reject correct responses due to spurious differences like formatting, punctuation, or valid alternative phrasings.

每个评估提示都应与可验证的响应或结果配对。你的验证器可以很简单，比如在基准事实和抽样响应之间进行精确的字符串比较，也可以很先进，比如让Claude来评判响应。要避免过于严格的验证器，这类验证器会因为格式、标点或有效的替代表达方式等无关差异而拒绝正确的响应。

For each prompt-response pair, you can optionally also specify the tools you expect an agent to call in solving the task, to measure whether or not agents are successful in grasping each tool's purpose during evaluation. However, because there might be multiple valid paths to solving tasks correctly, try to avoid overspecifying or overfitting to strategies.

对于每个提示-响应对，你还可以选择性地指定你期望智能体在解决任务时调用的工具，以衡量智能体在评估过程中是否成功理解了每个工具的用途。不过，由于正确解决任务可能存在多种有效途径，因此要尽量避免过度指定或过度拟合策略。

Running the evaluation 执行评估

We recommend running your evaluation programmatically with direct LLM API calls. Use simple agentic loops (`while` -loops wrapping alternating LLM API and tool calls): one loop for each evaluation task. Each evaluation agent should be given a single task prompt and your tools.

我们建议通过直接调用大语言模型API以编程方式进行评估。使用简单的智能体循环（用 `while` 循环包裹交替的大语言模型API调用和工具调用）：每个评估任务对应一个循环。每个评估智能体应被赋予一个单独的任务提示和你的工具。

In your evaluation agents' system prompts, we recommend instructing agents to output not just structured response blocks (for verification), but also reasoning and feedback blocks. Instructing agents to output these *before* tool call and response blocks may increase LLMs' effective intelligence by triggering chain-of-thought (CoT) behaviors.

在你的评估智能体系统提示中，我们建议指示智能体不仅输出结构化的响应块（用于验证），还要输出推理和反馈块。指示智能体在工具调用和响应块之*前*输出这些内容，可能会通过触发思维链（CoT）行为来提高大语言模型的有效智能。

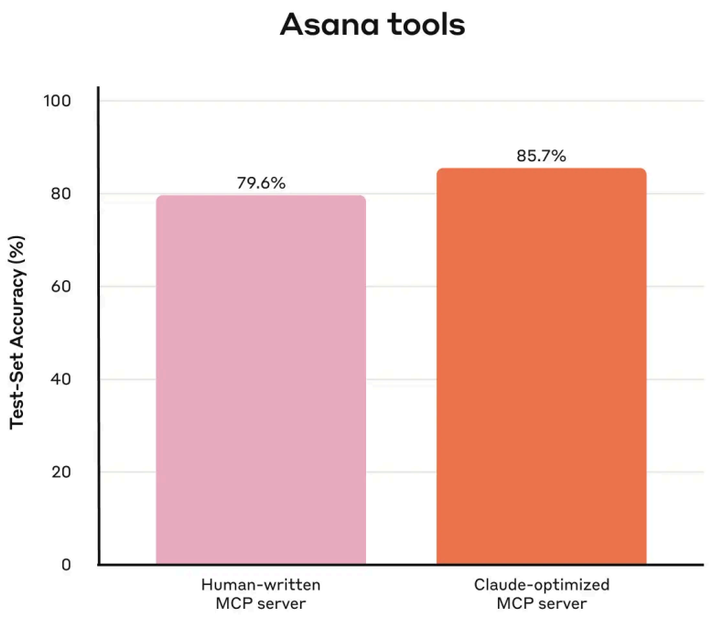
If you're running your evaluation with Claude, you can turn on interleaved thinking for similar functionality "off-the-shelf". This will help you probe why agents do or don't call certain tools and highlight specific areas of improvement

in tool descriptions and specs.

如果您使用Claude进行评估，可以开启交错思考功能，以获得类似的“现成”功能。这将帮助您探究智能体为什么调用或不调用某些工具，并突出显示工具描述和规格中需要改进的特定方面。

As well as top-level accuracy, we recommend collecting other metrics like the total runtime of individual tool calls and tasks, the total number of tool calls, the total token consumption, and tool errors. Tracking tool calls can help reveal common workflows that agents pursue and offer some opportunities for tools to consolidate.

除了顶级准确性之外，我们建议收集其他指标，例如单个工具调用和任务的总运行时间、工具调用的总数、总令牌消耗以及工具错误。跟踪工具调用有助于揭示智能体所采用的常见工作流程，并为工具整合提供一些机会。



Held-out test set performance of our internal Asana tools

Asana

Analyzing results 分析结果

Agents are your helpful partners in spotting issues and providing feedback on everything from contradictory tool descriptions to inefficient tool implementations and confusing tool schemas. However, keep in mind that what agents omit in their feedback and responses can often be more important than what they include. LLMs don’t always say what they mean.

智能体是您得力的合作伙伴，能帮助您发现各种问题并提供反馈，无论是矛盾的工具描述、低效的工具实现，还是令人困惑的工具架构。不过，请记住，智能体在反馈和回应中遗漏的内容往往比它们包含的内容更重要。大语言模型并不总是说出它们的真实想法。

Observe where your agents get stumped or confused. Read through your evaluation agents’ reasoning and feedback (or CoT) to identify rough edges.

Review the raw transcripts (including tool calls and tool responses) to catch any behavior not explicitly described in the agent's CoT. Read between the lines; remember that your evaluation agents don't necessarily know the correct answers and strategies.

观察你的智能体在哪些地方遇到困难或感到困惑。仔细阅读评估智能体的推理和反馈（或思维链），找出不足之处。查看原始记录（包括工具调用和工具响应），以发现智能体思维链中未明确描述的任何行为。留意字里行间的信息；记住，你的评估智能体不一定知道正确的答案和策略。

Analyze your tool calling metrics. Lots of redundant tool calls might suggest some rightsizing of pagination or token limit parameters is warranted; lots of tool errors for invalid parameters might suggest tools could use clearer descriptions or better examples. When we launched Claude's web search tool, we identified that Claude was needlessly appending `2025` to the tool's `query` parameter, biasing search results and degrading performance (we steered Claude in the right direction by improving the tool description).

分析你的工具调用指标。大量冗余的工具调用可能表明需要调整分页或令牌限制参数的大小；大量因参数无效导致的工具错误可能意味着工具需要更清晰的描述或更好的示例。当我们推出Claude的网络搜索工具时，我们发现Claude不必要地在该工具的 `query` 参数后附加了 `2025`，这会使搜索结果产生偏差并降低性能（我们通过改进工具描述引导Claude走向了正确的方向）。

Collaborating with agents

You can even let agents analyze your results and improve your tools for you. Simply concatenate the transcripts from your evaluation agents and paste them into Claude Code. Claude is an expert at analyzing transcripts and refactoring lots of tools all at once—for example, to ensure tool implementations and descriptions remain self-consistent when new changes are made.

你甚至可以让智能体分析你的结果并为你改进工具。只需将评估智能体的 transcripts 拼接起来，粘贴到 Claude Code 中即可。Claude 是分析 transcripts 并同时重构大量工具的专家——例如，确保在进行新更改时，工具的实现和描述保持自洽。

In fact, most of the advice in this post came from repeatedly optimizing our internal tool implementations with Claude Code. Our evaluations were created on top of our internal workspace, mirroring the complexity of our internal workflows, including real projects, documents, and messages.

事实上，这篇文章中的大部分建议都来自于我们借助Claude Code对内部工具实现进行的反复优化。我们的评估是在内部工作区基础上开展的，模拟了内部工作流程的复杂性，其中包括真实的项目、文档和消息。

We relied on held-out test sets to ensure we did not overfit to our "training" evaluations. These test sets revealed that we could extract additional performance improvements even beyond what we achieved with "expert" tool implementations—whether those tools were manually written by our researchers

or generated by Claude itself.

我们依靠留出的测试集来确保不会过拟合我们的“训练”评估。这些测试集表明，即使超出我们通过“专家级”工具实现所取得的成果，我们仍能进一步提升性能——无论这些工具是由我们的研究人员手动编写的，还是由Claude自身生成的。

In the next section, we'll share some of what we learned from this process.

在下一节中，我们将分享从这个过程中学到的一些东西。

Principles for writing effective tools

In this section, we distill our learnings into a few guiding principles for writing effective tools.

在本节中，我们将所学内容提炼为几条编写高效工具的指导原则。

Choosing the right tools for agents

More tools don't always lead to better outcomes. A common error we've observed is tools that merely wrap existing software functionality or API endpoints—whether or not the tools are appropriate for agents. This is because agents have distinct "affordances" to traditional software—that is, they have different ways of perceiving the potential actions they can take with those tools

工具越多并不总能带来更好的结果。我们发现一个常见的错误是，有些工具仅仅是对现有软件功能或API端点进行了包装，而不管这些工具是否适合智能体。这是因为智能体与传统软件具有不同的"功能可供性"，也就是说，它们在感知使用这些工具所能采取的潜在行动方面存在不同的方式。

LLM agents have limited "context" (that is, there are limits to how much information they can process at once), whereas computer memory is cheap and abundant. Consider the task of searching for a contact in an address book. Traditional software programs can efficiently store and process a list of contacts one at a time, checking each one before moving on.

大语言模型智能体的"上下文"有限（也就是说，它们一次能处理的信息量存在限制），而计算机内存既廉价又充足。以在通讯录中查找联系人这项任务为例，传统软件程序能够高效地存储联系人列表，并逐一处理，检查完一个再处理下一个。

However, if an LLM agent uses a tool that returns ALL contacts and then has to read through each one token-by-token, it's wasting its limited context space on irrelevant information (imagine searching for a contact in your address book by reading each page from top-to-bottom—that is, via brute-force search). The better and more natural approach (for agents and humans alike) is to skip to the relevant page first (perhaps finding it alphabetically).

然而，如果大语言模型智能体使用的工具返回了所有联系人，然后不得不逐令牌地阅读每个联系人的信息，那么它就是在将有限的上下文空间浪费在无关信息上（想象一下，在你的通讯录中查找某个联系人时，你需要从上到下逐页阅读——也就是通过暴力搜索的方式）。对于智能体和人类来说，更好且更自然的方法是先跳转到相关页面（或许可以按字母顺序查找）。

We recommend building a few thoughtful tools targeting specific high-impact workflows, which match your evaluation tasks and scaling up from there. In the address book case, you might choose to implement a `search_contacts` or `message_contact` tool instead of a `list_contacts` tool.

我们建议围绕特定的高影响力工作流程构建一些精心设计的工具，这些工具应与你的评估任务相匹配，并在此基础上逐步扩展。在通讯录的例子中，你或许可以选择实现 `search_contacts` 或 `message_contact` 工具，而非 `list_contacts` 工具。

Tools can consolidate functionality, handling potentially *multiple* discrete operations (or API calls) under the hood. For example, tools can enrich tool responses with related metadata or handle frequently chained, multi-step tasks

in a single tool call.

工具可以整合功能，在后台处理可能多个独立操作（或API调用）。例如，工具可以通过相关元数据丰富工具响应，或者在一次工具调用中处理频繁链接的多步骤任务。

Here are some examples: 以下是一些示例：

- Instead of implementing a `list_users`, `list_events`, and `create_event` tools, consider implementing a `schedule_event` tool which finds availability and schedules an event.

  与其实现 `list_users`、`list_events` 和 `create_event` 工具，不如考虑实现一个 `schedule_event` 工具，该工具可以查找空闲时间并安排事件。

- Instead of implementing a `read_logs` tool, consider implementing a `search_logs` tool which only returns relevant log lines and some surrounding context.

  不必实现 `read_logs` 工具，而是考虑实现一个 `search_logs` 工具，该工具仅返回相关的日志行以及一些周边上下文。

- Instead of implementing `get_customer_by_id`, `list_transactions`, and `list_notes` tools, implement a `get_customer_context` tool which compiles all of a customer's recent & relevant information all at once.

  不要实现 `get_customer_by_id`、`list_transactions` 和 `list_notes` 这些工具，而是实现一个 `get_customer_context` 工具，该工具可以一次性汇总客户的所有近期相关信息。

Make sure each tool you build has a clear, distinct purpose. Tools should enable agents to subdivide and solve tasks in much the same way that a human would, given access to the same underlying resources, and simultaneously reduce the context that would have otherwise been consumed by intermediate outputs.

确保你构建的每个工具都有明确、独特的用途。工具应当能让智能体像人类一样，在获得相同基础资源的情况下，对任务进行细分和解决，同时减少原本会被中间输出占用的上下文。

Too many tools or overlapping tools can also distract agents from pursuing efficient strategies. Careful, selective planning of the tools you build (or don't build) can really pay off.

过多的工具或功能重叠的工具也会分散智能体的注意力，使其无法采用高效策略。谨慎且有选择地规划要构建（或不构建）的工具，确实能带来回报。

Namespacing your tools

Your AI agents will potentially gain access to dozens of MCP servers and hundreds of different tools–including those by other developers. When tools overlap in function or have a vague purpose, agents can get confused about which ones to use.

你的人工智能智能体可能会访问数十个MCP服务器和数百种不同的工具，包括其他开发者提供的工具。当工具在功能上重叠或用途模糊时，智能体可能会在选择使用哪一种工具时感到困惑。

Namespacing (grouping related tools under common prefixes) can help delineate boundaries between lots of tools; MCP clients sometimes do this by default. For example, namespacing tools by service (e.g., `asana_search`, `jira_search`) and by resource (e.g., `asana_projects_search`, `asana_users_search`), can help agents select the right tools at the right time.

命名空间（将相关工具归类到共同前缀下）有助于划分大量工具之间的界限；MCP客户端有时会默认采用这种方式。例如，按服务（如 `asana_search`、`jira_search`）和按资源（如 `asana_projects_search`、`asana_users_search`）对工具进行命名空间划分，能够帮助智能体在恰当的时机选择合适的工具。

We have found selecting between prefix- and suffix-based namespacing to have non-trivial effects on our tool-use evaluations. Effects vary by LLM and we encourage you to choose a naming scheme according to your own evaluations.

我们发现，在基于前缀和基于后缀的命名空间之间进行选择，会对我们的工具使用评估产生重要影响。不同的大语言模型（LLM）所受的影响各不相同，因此我们建议您根据自己的评估结果来选择命名方案。

Agents might call the wrong tools, call the right tools with the wrong parameters, call too few tools, or process tool responses incorrectly. By selectively implementing tools whose names reflect natural subdivisions of tasks, you simultaneously reduce the number of tools and tool descriptions loaded into the agent's context and offload agentic computation from the agent's context back into the tool calls themselves. This reduces an agent's overall risk of making mistakes.

智能体可能会调用错误的工具，使用错误的参数调用正确的工具，调用的工具数量过少，或者错误地处理工具的响应。通过有选择地使用那些名称能反映任务自然细分的工具，你可以同时减少加载到智能体语境中的工具和工具描述的数量，并将智能体的计算任务从其语境中转移到工具调用本身。这降低了智能体出错的总体风险。

Returning meaningful context from your tools

In the same vein, tool implementations should take care to return only high signal information back to agents. They should prioritize contextual relevance over flexibility, and eschew low-level technical identifiers (for example: `uuid`, `256px_image_url`, `mime_type`). Fields like `name`, `image_url`, and `file_type` are much more likely to directly inform agents' downstream actions and responses.

同样地，工具实现应注意只向智能体返回高信号信息。它们应优先考虑上下文相关性而非灵活性，并且避免使用低级技术标识符（例如：`uuid`、`256px_image_url`、`mime_type`）。像`name`、`image_url`和`file_type`这类字段更有可能直接为智能体的后续行动和响应提供信息。

Agents also tend to grapple with natural language names, terms, or identifiers significantly more successfully than they do with cryptic identifiers. We've found that merely resolving arbitrary alphanumeric UUIDs to more semantically meaningful and interpretable language (or even a 0-indexed ID scheme) significantly improves Claude's precision in retrieval tasks by reducing hallucinations.

智能体在处理自然语言名称、术语或标识符时，往往比处理晦涩的标识符要成功得多。我们发现，仅仅将任意的字母数字UUID转换为更具语义且易于理解的语言（甚至是0索引的ID方案），就能通过减少幻觉显著提高Claude在检索任务中的准确性。

In some instances, agents may require the flexibility to interact with both natural language and technical identifiers outputs, if only to trigger downstream tool calls (for example, `search_user(name='jane')` → `send_message(id=12345)`). You can enable both by exposing a simple `response_format` enum parameter in your tool, allowing your agent to control whether tools return "`concise`" or "`detailed`" responses (images below).

在某些情况下，智能体可能需要灵活地与自然语言和技术标识符输出进行交互，哪怕只是为了触发下游工具调用（例如，`search_user(name='jane')` → `send_message(id=12345)`）。你可以通过在工具中设置一个简单的`response_format`枚举参数来同时支持这两种方式，让智能体能够控制工具返回的是"`concise`"还是"`detailed`"的响应（见下图）。

You can add more formats for even greater flexibility, similar to GraphQL where you can choose exactly which pieces of information you want to receive. Here is an example ResponseFormat enum to control tool response verbosity:

你可以添加更多格式以获得更大的灵活性，这类似于GraphQL，你可以精确选择想要接收的信息片段。以下是一个用于控制工具响应详细程度的ResponseFormat枚举示例：

```
enum ResponseFormat {
   DETAILED = "detailed",
   CONCISE = "concise"
}
```

Copy

Here's an example of a detailed tool response (206 tokens):

以下是一个详细工具响应的示例（206个标记）：

```
● I'll search slack for recent bug reports and use the detailed format to see which channel IDs and threads to
investigate further.

● slack - search (MCP)(query: "bug", sort: "timestamp", sortDir: "desc", limit: 100, responseFormat:
"detailed")
 L Search results for: "bug"

    ═══ Result 1 of 89 ═══
    Channel: #dev (C1234567890)
    From: @jane.doe (U123456789)
    Time: 2024-01-15 10:30:45 UTC
    TS: 1705316445.123456
    Text: Found a critical bug in the login flow.

    ═══ Result 2 of 89 ═══
    Channel: DM with @john.smith
    From: @john.smith (U987654321)
    Time: 2024-01-14 15:22:18 UTC
    TS: 1705247738.234567
    Text: The bug report for issue #123 is ready for review
    Files: bug-report-123.pdf
    ...
```

Here's an example of a concise tool response (72 tokens):

以下是一个简洁的工具响应示例（72个令牌）：

```
● I'll search slack for recent bug reports and use the concise format to read as many messages as possible.

● slack - search (MCP)(query: "bug", sort: "timestamp", sortDir: "desc", limit: 100, responseFormat: "concise")
 L Search: "bug" (89 results)

    1. #dev - @jane.doe: Found a critical bug in the login flow. [Jan 15]
    2. DM - @john.smith: The bug report for issue #123 is ready for review [Jan 14]
    ...
```

Slack threads and thread replies are identified by unique `thread_ts` which are required to fetch thread replies. `thread_ts` and other IDs (`channel_id`, `user_id`) can be retrieved from a "`detailed`" tool response to enable further tool calls that require these. "`concise`" tool responses return only thread content and exclude IDs. In this example, we use ~⅓ of the tokens with "`concise`" tool responses.

Slack thread_ts thread_ts ID channel_id user_id "detailed" "concise" ID "concise"

Even your tool response structure—for example XML, JSON, or Markdown—can have an impact on evaluation performance: there is no one-size-fits-all solution. This is because LLMs are trained on next-token prediction and tend to perform

better with formats that match their training data. The optimal response structure will vary widely by task and agent. We encourage you to select the best response structure based on your own evaluation.

甚至连你的工具响应结构（例如XML、JSON或Markdown）都可能对评估性能产生影响：不存在放之四海而皆准的解决方案。这是因为大型语言模型（LLMs）是通过预测下一个token来训练的，并且在处理与训练数据匹配的格式时往往表现更佳。最佳的响应结构会因任务和智能体的不同而有很大差异。我们建议你根据自己的评估来选择最佳的响应结构。

Optimizing tool responses for token efficiency

Optimizing the quality of context is important. But so is optimizing the *quantity* of context returned back to agents in tool responses.

优化上下文质量很重要。但优化工具响应中返回给智能体的上下文数量也同样重要。

We suggest implementing some combination of pagination, range selection, filtering, and/or truncation with sensible default parameter values for any tool responses that could use up lots of context. For Claude Code, we restrict tool responses to 25,000 tokens by default. We expect the effective context length of agents to grow over time, but the need for context-efficient tools to remain.

对于任何可能占用大量上下文的工具响应，我们建议结合使用分页、范围选择、筛选和/或截断功能，并设置合理的默认参数值。对于Claude Code，我们默认将工具响应限制在25,000个令牌。我们预计智能体的有效上下文长度会随着时间的推移而增长，但对上下文高效工具的需求仍将存在。

If you choose to truncate responses, be sure to steer agents with helpful instructions. You can directly encourage agents to pursue more token-efficient strategies, like making many small and targeted searches instead of a single, broad search for a knowledge retrieval task. Similarly, if a tool call raises an error (for example, during input validation), you can prompt-engineer your error responses to clearly communicate specific and actionable improvements, rather than opaque error codes or tracebacks.

如果您选择截断响应，请务必用有用的指令引导智能体。您可以直接鼓励智能体采用更节省token的策略，例如，在知识检索任务中，进行多次小型且有针对性的搜索，而不是一次宽泛的搜索。同样，如果工具调用出现错误（例如，在输入验证期间），您可以通过提示工程来设计错误响应，以清晰传达具体且可操作的改进建议，而非模糊的错误代码或回溯信息。

Here’s an example of a truncated tool response:

以下是一个截断的工具响应示例：

```
● I'll find all of your transactions on Stripe and provide a summary for you.

● stripe - transactions_search (MCP)(limit: 5000, responseFormat: "concise")
 L ## Transaction Search Results

    Found **2,847 transactions** matching your query.

    The results are truncated. Showing first 3 results:

    | Date | Description | Amount | Category |
    ├──────┼─────────────┼────────┼──────────┤
    | 2024-01-15 | Payment from Acme Corp | +$5,200.00 | Revenue |
    | 2024-01-14 | Payment from TechStart | +$3,100.00 | Revenue |
    | 2024-01-13 | Payment from Cloud Co. | +$3,100.00 | Revenue |

    **Summary of all 2,847 results:**
    - Total Revenue: $458,291.00
    - Date Range: Jan 1 - Jan 15, 2024

    ## To refine these results, you can:

    - **Search for specific vendors**: Use `transactions_search(payee: "Acme Corp")` to see only Acme Corp
charges
    - **Filter by amount range**: Use `transactions_search(minAmount: 1000, maxAmount: 5000)`
    - **Get next page**: Use `transactions_search(query: <query>, page: 2)`
```

Here's an example of an unhelpful error response:

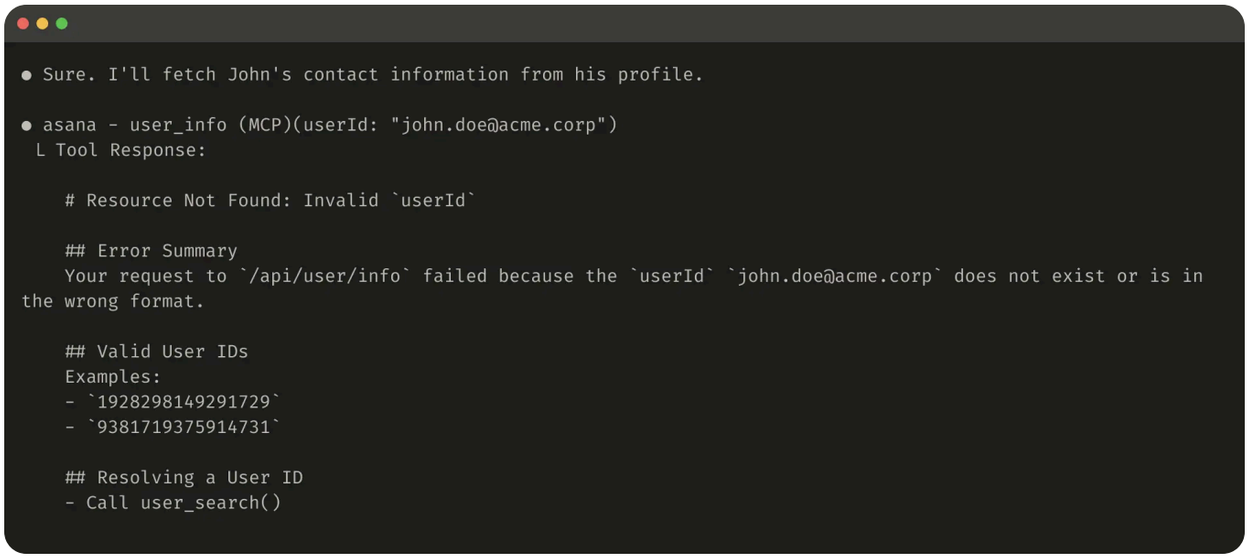
以下是一个无用的错误响应示例：

```
● Sure. I'll fetch John's contact information from his profile.

● asana - user_info (MCP)(userId: "john.doe@acme.corp")
 L {
     "error": {
       "code": "RESOURCE_NOT_FOUND",
       "status": 422,
       "message": "Invalid value",
       "details": {
         "field": "userId",
         "value": "john.doe@acme.corp",
         "type": "invalid_value"
       }
     }
   }
```

Here's an example of a helpful error response:

以下是一个有用的错误响应示例：



Tool truncation and error responses can steer agents towards more token-efficient tool-use behaviors (using filters or pagination) or give examples of correctly formatted tool inputs.

Prompt-engineering your tool descriptions

We now come to one of the most effective methods for improving tools: prompt-engineering your tool descriptions and specs. Because these are loaded into your agents' context, they can collectively steer agents toward effective tool-calling behaviors.

现在我们来介绍一种改进工具的最有效方法之一：对工具描述和规格进行提示词工程。由于这些内容会被加载到智能体的上下文环境中，它们能够共同引导智能体做出有效的工具调用行为。

When writing tool descriptions and specs, think of how you would describe your tool to a new hire on your team. Consider the context that you might implicitly bring—specialized query formats, definitions of niche terminology, relationships between underlying resources—and make it explicit. Avoid ambiguity by clearly describing (and enforcing with strict data models) expected inputs and outputs. In particular, input parameters should be unambiguously named: instead of a parameter named `user`, try a parameter named `user_id`.

在撰写工具描述和规格时，要思考你会如何向团队的新员工介绍这个工具。考虑到你可能会隐含提及的背景信息——专业的查询格式、特定术语的定义、基础资源之间的关系——并将这些信息明确表述出来。通过清晰描述（并借助严格的数据模型来确保）预期的输入和输出，避免歧义。特别是输入参数的命名应当清晰明确：不要使用名为 `user` 的参数，而应尝试使用 `user_id` 这样的参数名称。

With your evaluation you can measure the impact of your prompt engineering with greater confidence. Even small refinements to tool descriptions can yield dramatic improvements. Claude Sonnet 3.5 achieved state-of-the-art performance on the SWE-bench Verified evaluation after we made precise refinements to tool descriptions, dramatically reducing error rates and improving task completion.

通过评估，您可以更有信心地衡量提示词工程的影响。即使对工具描述进行微小的改进，也能带来显著的提升。在我们对工具描述进行精确优化后，Claude Sonnet 3.5在SWE-bench Verified评估中取得了最先进的性能，大幅降低了错误率并提高了任务完成度。

You can find other best practices for tool definitions in our Developer Guide. If you're building tools for Claude, we also recommend reading about how tools are dynamically loaded into Claude's system prompt. Lastly, if you're writing tools for an MCP server, tool annotations help disclose which tools require open-world access or make destructive changes.

你可以在我们的《开发者指南》中找到工具定义的其他最佳实践。如果你正在为Claude构建工具，我们还建议你了解工具如何动态加载到Claude的"系统提示词"中。最后，如果你正在为MCP服务器编写工具，"工具注释"有助于披露哪些工具需要开放世界访问权限或会做出破坏性更改。

Looking ahead

To build effective tools for agents, we need to re-orient our software development practices from predictable, deterministic patterns to non-deterministic ones.

要为智能体打造高效的工具，我们需要将软件开发实践从可预测、确定性的模式转向非确定性模式。

Through the iterative, evaluation-driven process we've described in this post, we've identified consistent patterns in what makes tools successful: Effective tools are intentionally and clearly defined, use agent context judiciously, can be combined together in diverse workflows, and enable agents to intuitively solve real-world tasks.

通过我们在本文中描述的这种迭代式、以评估为导向的流程，我们发现了让工具获得成功的一些一致模式：有效的工具具有明确且经过精心设计的定义，能审慎地利用智能体的上下文信息，可以在各种工作流中组合使用，并且能让智能体直观地解决现实世界中的任务。

In the future, we expect the specific mechanisms through which agents interact with the world to evolve—from updates to the MCP protocol to upgrades to the underlying LLMs themselves. With a systematic, evaluation-driven approach to improving tools for agents, we can ensure that as agents become more capable, the tools they use will evolve alongside them.

未来，我们预计智能体与世界交互的具体机制将会不断发展——从MCP协议的更新到基础大语言模型本身的升级。通过一种系统化、以评估为导向的方法来改进智能体工具，我们可以确保随着智能体能力的提升，它们所使用的工具也能随之发展。

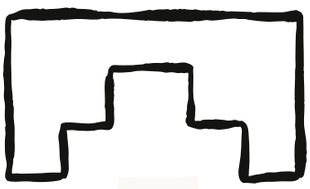
Acknowledgements

Written by Ken Aizawa with valuable contributions from colleagues across Research (Barry Zhang, Zachary Witten, Daniel Jiang, Sami Al-Sheikh, Matt Bell, Maggie Vo), MCP (Theodora Chu, John Welsh, David Soria Parra, Adam Jones), Product Engineering (Santiago Seira), Marketing (Molly Vorwerck), Design (Drew Roper), and Applied AI (Christian Ryan, Alexander Bricken).

本文由Ken Aizawa撰写，来自研究部门（Barry Zhang、Zachary Witten、Daniel Jiang、Sami Al-Sheikh、Matt Bell、Maggie Vo）、MCP（Theodora Chu、John Welsh、David Soria Parra、Adam Jones）、产品工程部门（Santiago Seira）、市场部门（Molly Vorwerck）、设计部门（Drew Roper）以及应用AI部门（Christian Ryan、Alexander Bricken）的同事也提供了宝贵贡献。

[1]Beyond training the underlying LLMs themselves.

1除了训练底层的大语言模型本身之外。



Academy. Earn certificates upon completion.

通过Anthropic学院的课程，掌握API开发、模型上下文协议和Claude代码。完成课程后可获得证书。

Explore courses

Get the developer newsletter

Product updates, how-tos, community spotlights, and more. Delivered monthly to your inbox.

产品更新、使用指南、社区焦点等内容。每月发送至您的收件箱。

Enter your email

Please provide your email address if you'd like to receive our monthly developer newsletter. You can unsubscribe at any time.