Agent 技术分享

如何让大模型理解和交互代码仓库

BITSE

2025-06-23

Powered by Typst & Touying

Outline

引言

知名评测集 **SWE-bench Verified** (SoftWare Engineering BENCHmark)

- SWE-bench 从 GitHub 开源项目的真实 Issue 中提取代码问题，要求模型理解上下文、分析需求，并生成可运行的 patch
- 与传统的代码生成 benchmark 不同的是，SWE-bench 聚焦评测 **真是软件开发工程能力**
- 近半年内已成为最受关注的代码评测指标，出现在所有主流大模型的技术报告中
- **理解代码仓库**，**修改相应源码**，**执行测试获得反馈**，LLM 是如何做到的?

Agent 技术

1. 模型侧关键技术 —— Function Calling
2. Agent 基本原理和工作模式
3. 代码 Agent 是怎么实现的 —— 以 SWE-Agent 和 Anthropic 为例

纯 LLM 的能力和局限

现代 LLM 能够很好地完成几乎所有“文本 → 文本”任务，但是距离完成真实任务还存在以下局限：

1. 知识固定

- LLM 在预训练阶段获得通用知识，固定不变，对于频繁变化的、私有的内容无能为力
  - 例：北京下周的天气预报；用户想编辑的代码库文档库

2. 光说不做

- LLM 只能输出文本，不能实际采取行动，依赖人工开发外围应用

3. 模式受限：在传统提示词模式下，LLM 只能按照人类预定义方式工作，但大量真实世界任务无法事先定义工作流 (workflow)

在这样的背景下，基于工具调用的 Agent 技术应运而生

例子

例 1 ChatGPT 网页版中的工具调用（未在网页 UI 上勾选 search skill）

GPT 理解了我的需求并主动从权威信息渠道获取了天气预报，最终给出答案

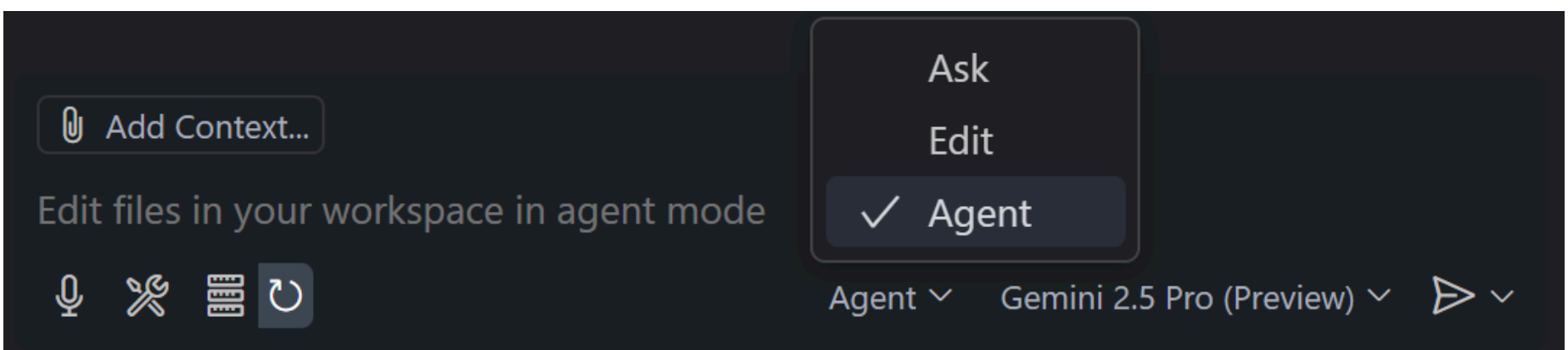
例子

例 2 开发环境中的编码 Agent

24 年末至 25 年初，各大厂商陆续推出 AI IDE 中的 Agent 模式（更加自主的编码助手）

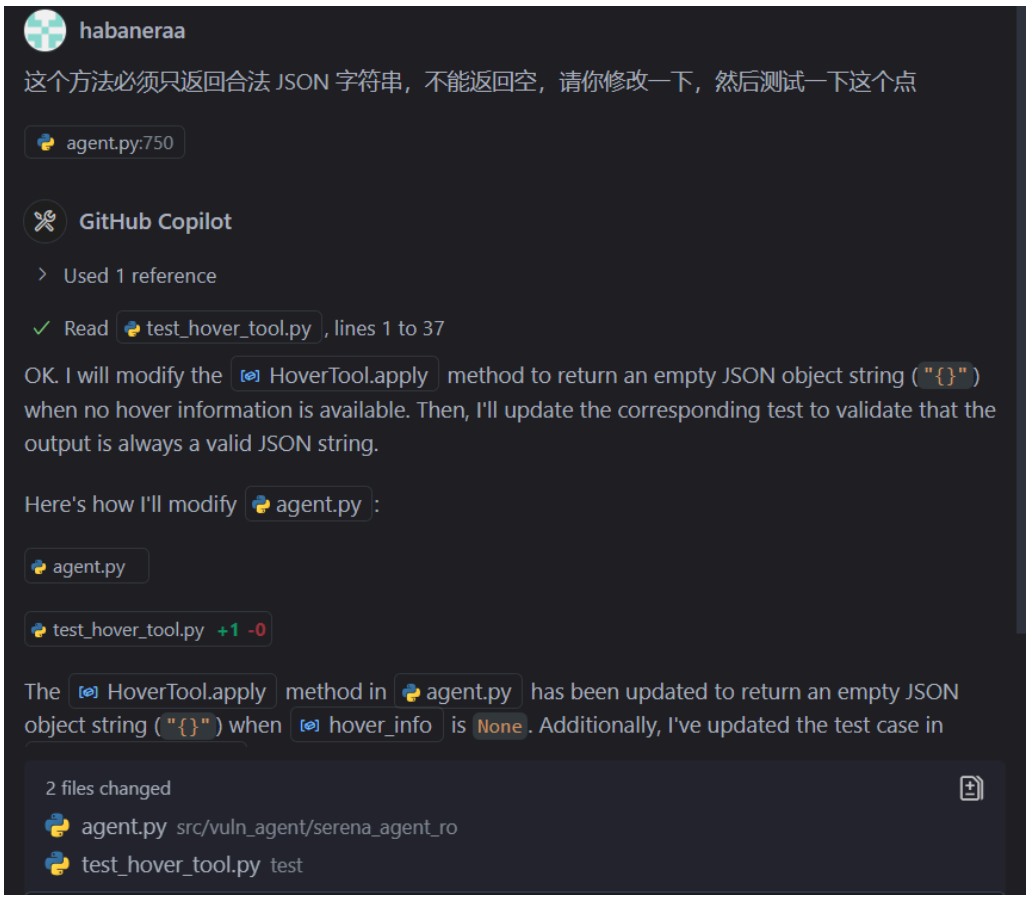
- 包括： Cursor, Windsurf, GitHub Copilot (VSCode), 字节 Trae

例如 VSCode 开启 Copilot 插件后可以打开这样一个页面：



图标表示该 Agent Mode 可以访问一系列工具，并自主完成用户的要求

例子


habaneraa
这个方法必须只返回合法 JSON 字符串，不能返回空，请你修改一下，然后测试一下这个点
agent.py:750
GitHub Copilot
Used 1 reference
Read test_hover_tool.py, lines 1 to 37
OK. I will modify the HoverTool.apply method to return an empty JSON object string ("{}") when no hover information is available. Then, I'll update the corresponding test to validate that the output is always a valid JSON string.
Here's how I'll modify agent.py:
agent.py
test_hover_tool.py +1 -0
The HoverTool.apply method in agent.py has been updated to return an empty JSON object string ("{}") when hover_info is None. Additionally, I've updated the test case in
2 files changed
agent.py src/vuln_agent/serena_agent_ro
test_hover_tool.py test

Outline

概念

Function Calling 或称 Tool Calling 是一种让 LLM 与外部系统交互的机制

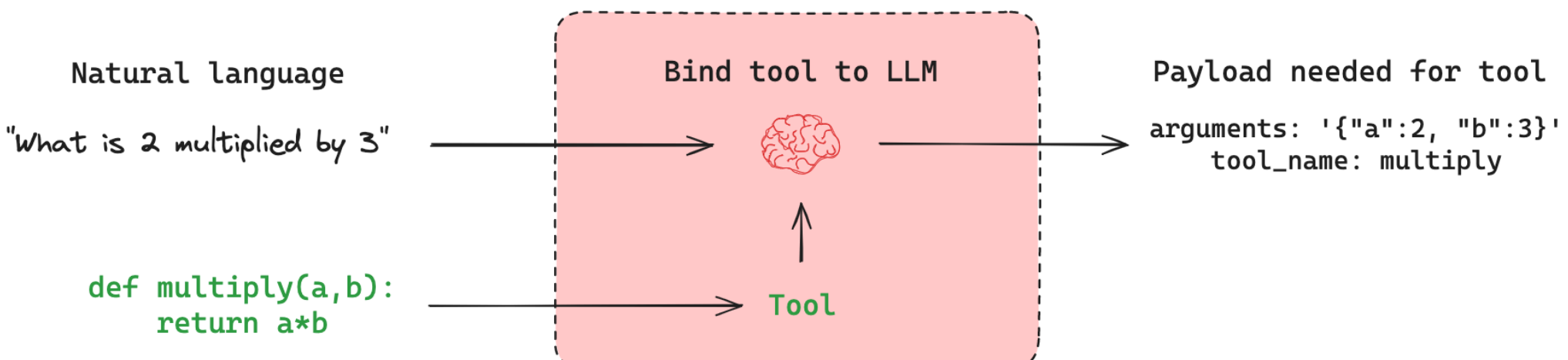
1. LLM 使用纯文本作为输入输出, 为了在 LLM 和外部系统之间建立桥梁, 使用 “function” 的概念
   - 例如，互联网搜索是一种典型的外部系统，用于给 LLM 提供即时信息
   - 为了让 LLM 和它交互，我们可以定义出一种函数接口提供给 LLM
   - ```python
     def search_google(query_string: str) -> list[str]:
     ```
2. 结构化输出
   - 在一次 LLM 请求中，请求方需要提供可用工具的 spec
     - 通常而言，可用工具的 binding 完全在 prompt 中
   - LLM 若决定使用工具，必定生成符合 spec 的结构化输出 (JSON)
     - 对于 LLM 侧而言，这一般是通过微调 + constrained decoding 实现的
     - **保证百分百可靠**
     ```

Tool 的创建与绑定

通常而言，函数的具体实现由客户端而非 LLM 服务端提供

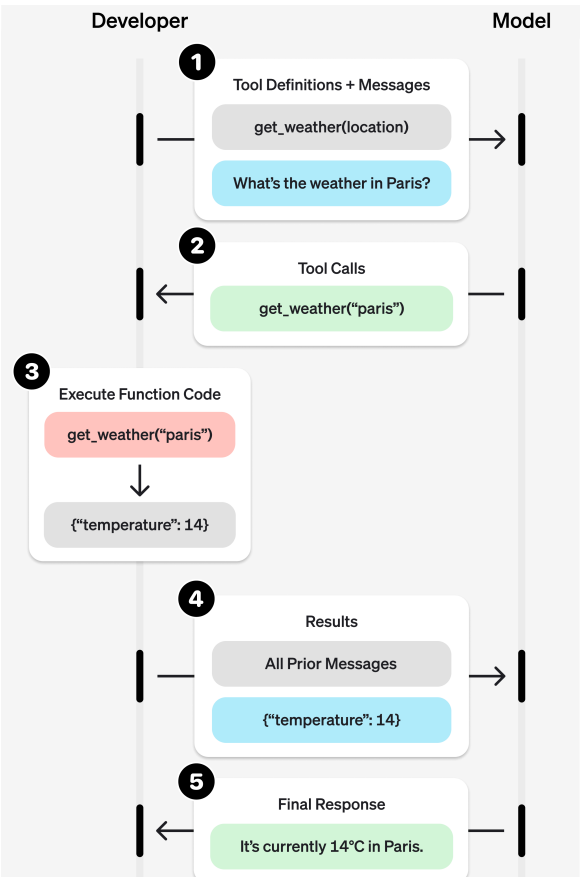
OpenAI Platform 也会提供一些内置工具集，但这属于额外功能

在一次 LLM 请求中，需要向 LLM 提供接口定义+自然语言描述，这种标准化格式会被 LLM 后端服务转化为纯 prompt

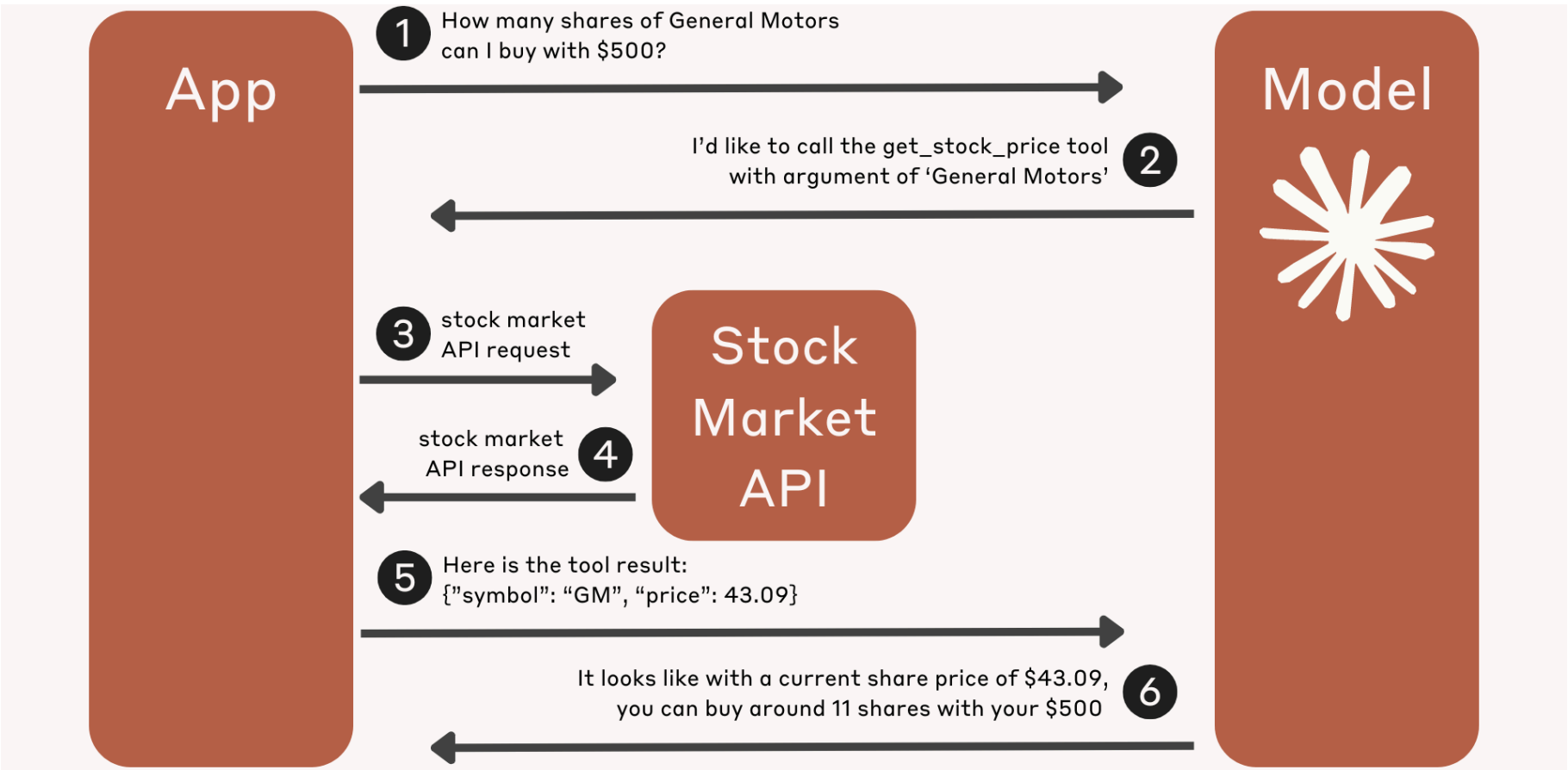


Tool 的创建与绑定

```json
{
  "type": "function",
  "name": "get_weather",
  "description": "获取指定地理位置的温度",
  "parameters": {  // 定义键值对参数列表
    "type": "object",
    "properties": {
      "location": {
        "type": "string",  // 指定参数类型
        "description": "城市名例如 paris"
      }
    },
    "required": ["location"],
  }
}
```



Tool Use 流程


App
Model
1
How many shares of General Motors can I buy with $500?
I'd like to call the get_stock_price tool with argument of 'General Motors'
2
3
stock market API request
Stock Market API
stock market API response
4
5
Here is the tool result: {"symbol": "GM", "price": 43.09}
It looks like with a current share price of $43.09, you can buy around 11 shares with your $500
6

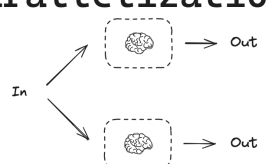
Outline

Agent 概念

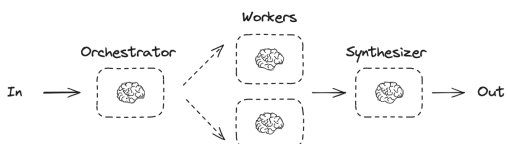
Workflows

Prompt Chaining

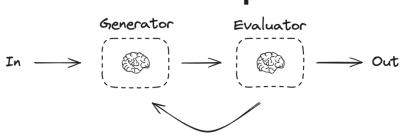
Parallelization



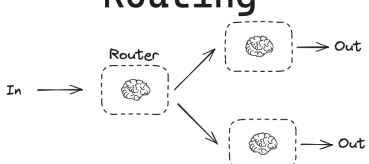
Orchestrator-Worker



Evaluator-optimizer



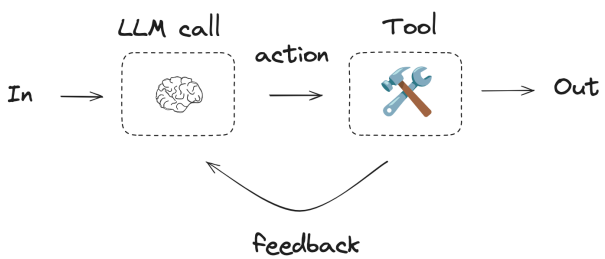
Routing



LLM is embedded in predefined code paths

LLM directs control flow through predefined code paths

Agent



LLM directs its own actions based on environmental feedback

Agent 概念

狭义 Agent 的概念解释：

Workflows are systems where LLMs and tools are orchestrated through predefined code paths. Agents, on the other hand, are systems where LLMs dynamically direct their own processes and tool usage, maintaining control over how they accomplish tasks.

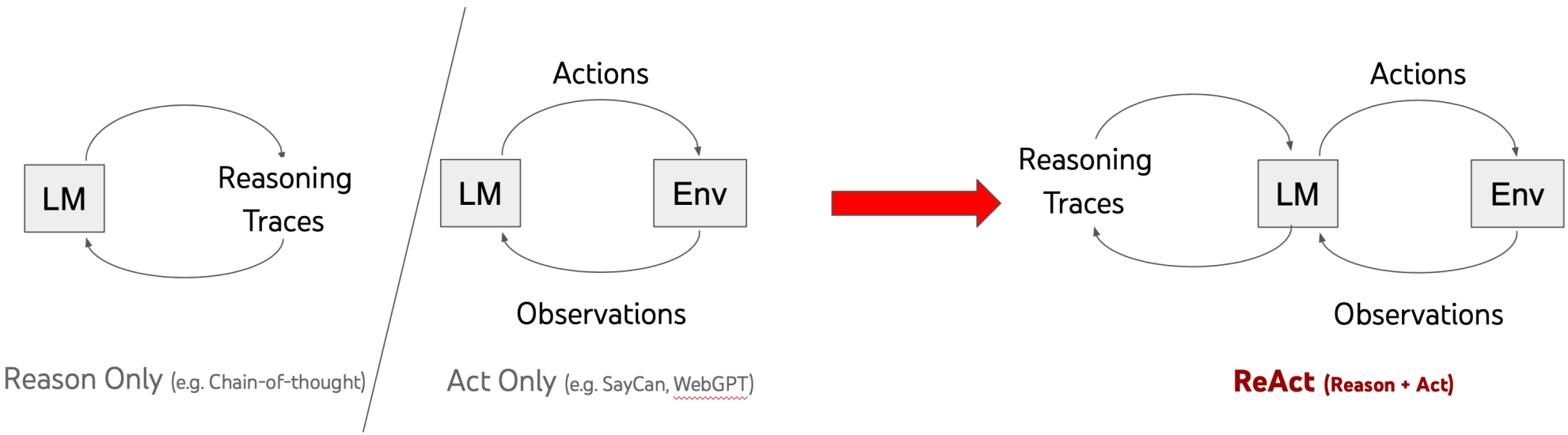
agent 不仅能做，它还能自主决策做什么，这个决策是通过 LLM 来完成的

An AI agent is a system that uses an LLM to decide the control flow of an application.

比如：

- DeepSeek 网页版开启 “联网搜索”，是一种预定义的工作流，模型并没有做自主决策
- VSCode 里的 Copilot Agent 有一定自主规划和行动的能力

ReAct 模式



[ICLR 2023] ReAct: Synergizing Reasoning and Acting in Language Models

Agent 设计的早期代表性工作

ReAct 模式

在 Function Calling 技术成熟前，ReAct 模式是三步骤循环：

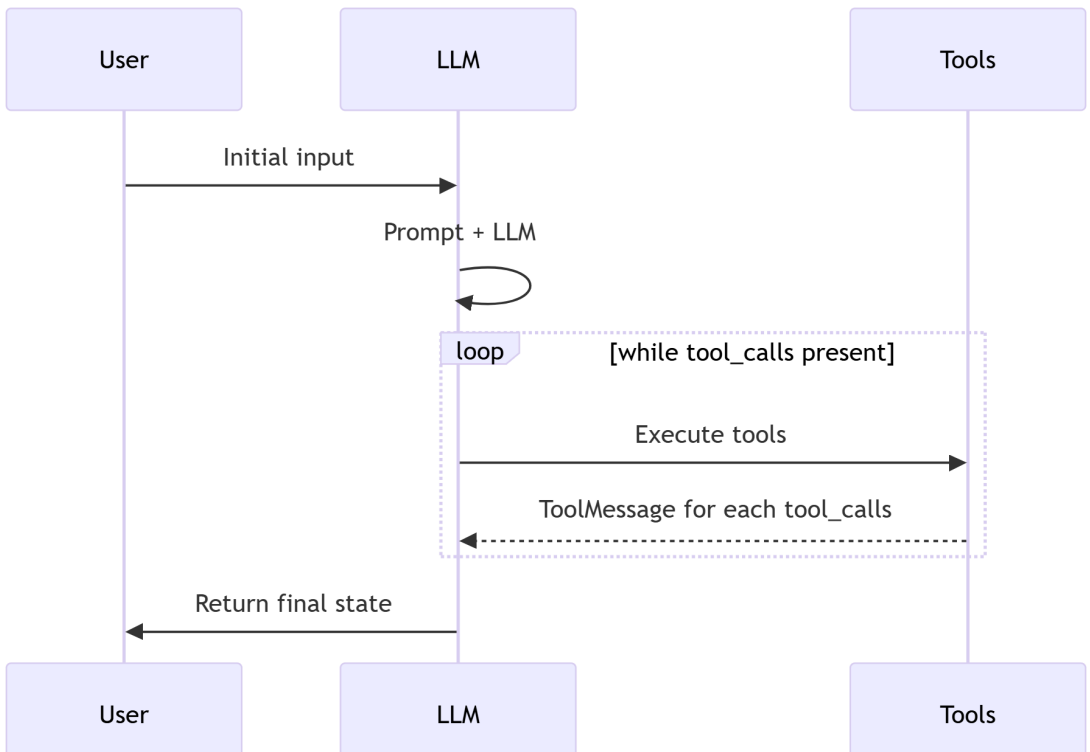
1. Thought - 根据前文，大模型生成思维链分析当前的情况
2. Action - 大模型生成结构化行为
3. Observation - 外部系统执行行为，将结果添加回大模型上下文

在 Function Calling 下，Thought 即大模型一次调用中的 message content， Action 即大模型生成的 tool use，而 Observation 会作为 tool message 放进 chat history 中

ReAct 模式

```text
<USER> What's the weather in London ?
<AI>
Thought: To answer the question, I need to get the current weather in London.
Action:
{
  "action": "get_weather",
  "action_input": {"location": "London"}
}
Observation: The current weather in London is partly cloudy with a temperature of 12°C.
Thought: I now know the final answer.
Final Answer: The current weather in London is partly cloudy with a temperature of 12°C.
```

ReAct 模式


User
LLM
Tools
Initial input
Prompt + LLM
loop
[while tool_calls present]
Execute tools
ToolMessage for each tool_calls
Return final state
User
LLM
Tools

Outline

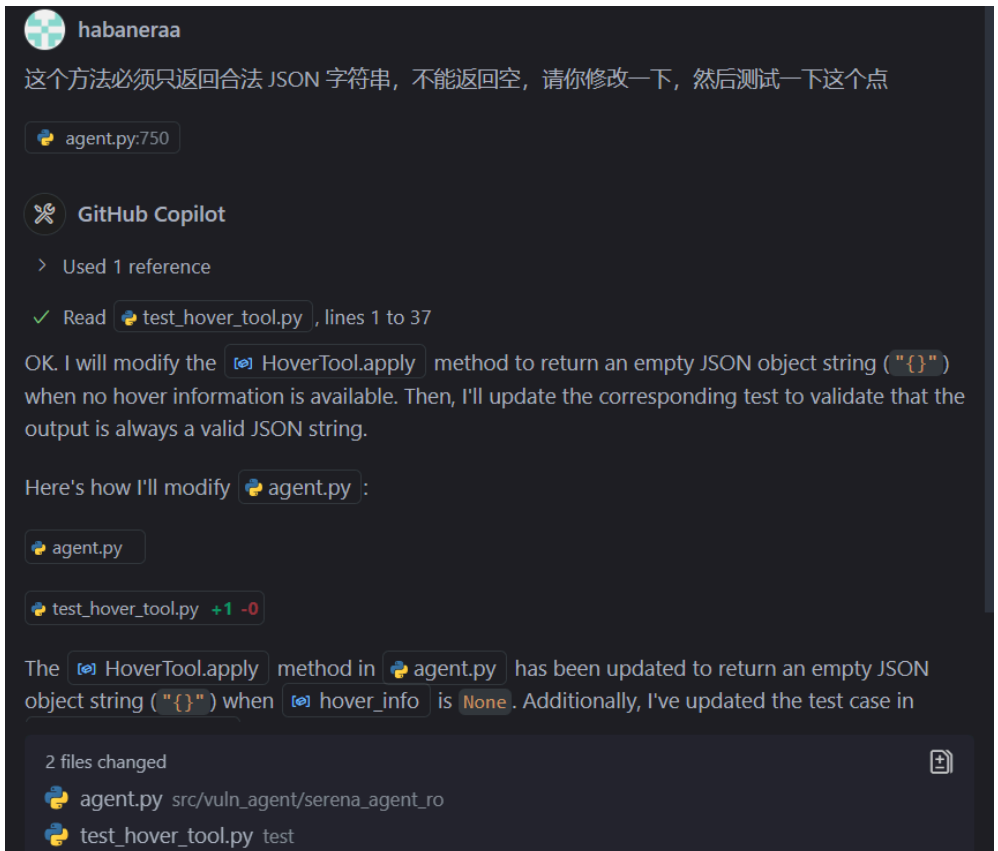
Coding Agent 的能力展示

可以看到 Copilot Agent 收到用户需求后：

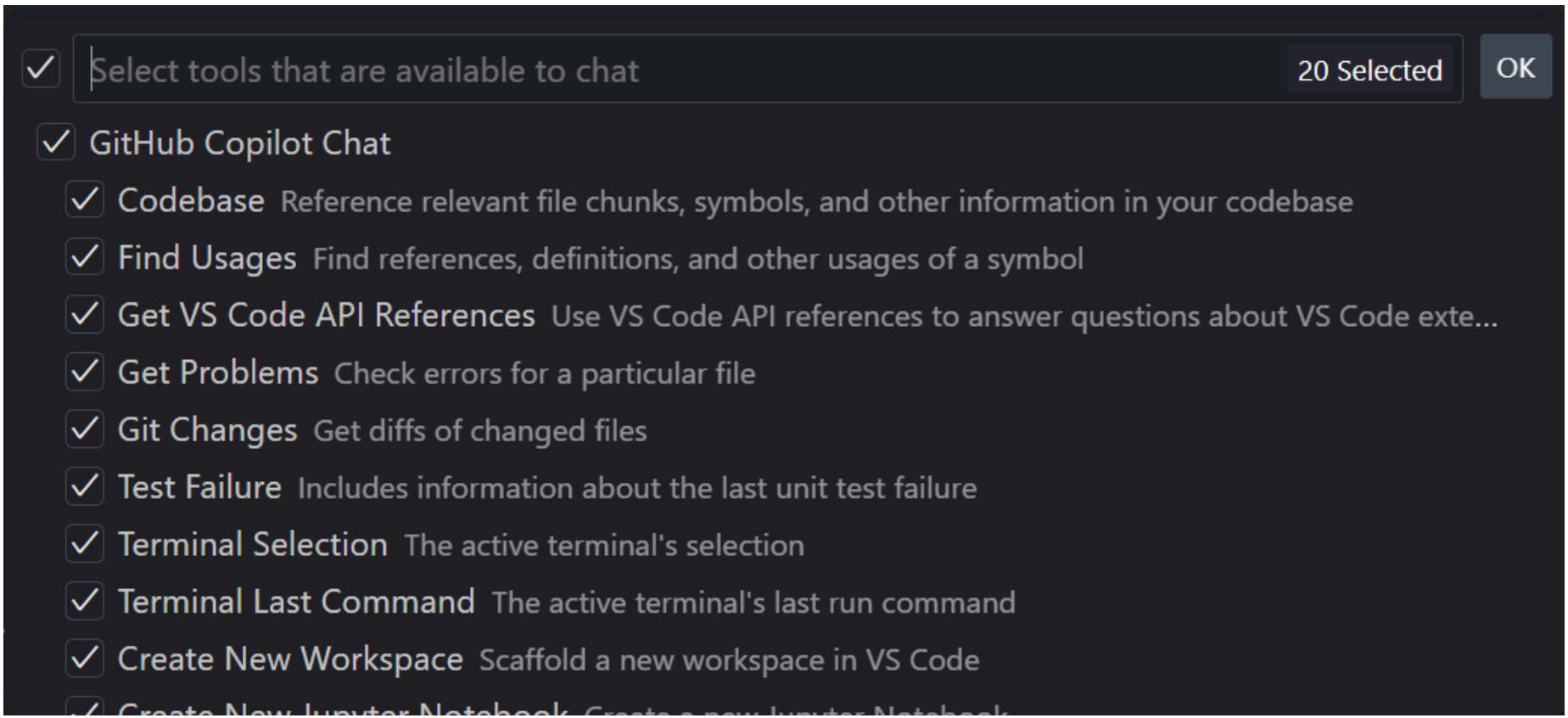
- 主动阅读了另一个源码文件的 37 行
- 并且产生了两处代码修改
- 最终向用户解释了代码改动

从应用层面看，这是一种自主理解代码仓库并产生修改的能力

软件工程任务各种各样，很难事先定义完善的工作流 (workflow)，而 coding agent 可以根据实际情况决定读代码/编辑代码/执行测试/查看报错/运行终端等操作，通过内部循环不断尝试，直到用户任务被完成或超出尝试限制



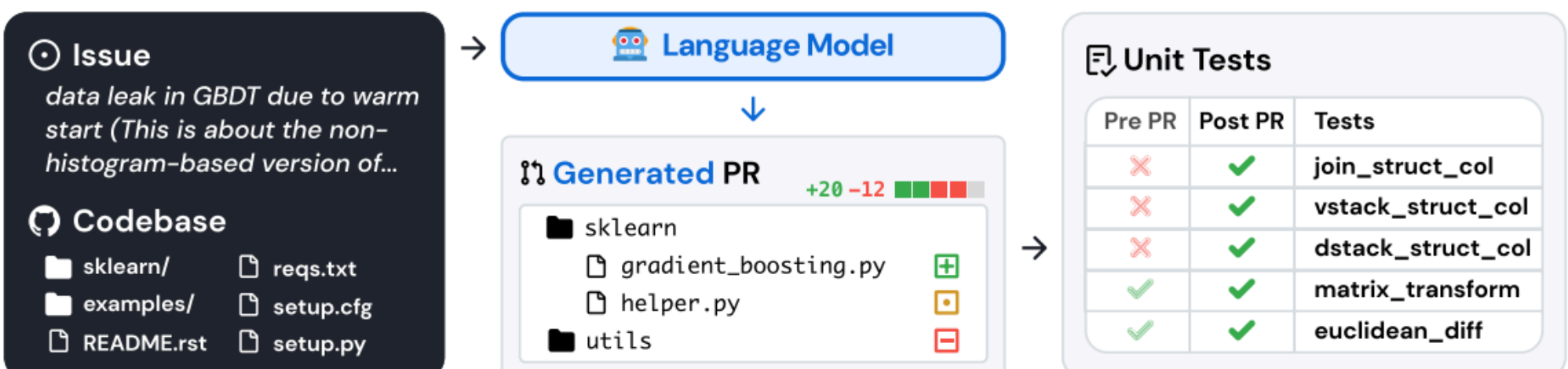
Coding Agent 的能力展示



SWE-bench 和 SWE-Agent

SWE-bench: 真实软件工程问题 benchmark

- 来自 ICLR 2024 研究 SWE-bench: Can Language Models Resolve Real-World GitHub Issues?
- 包含来自 12 个 Python 仓库的 2,294 个代码任务
- AI 需要根据问题描述直接修改整个代码库，然后使用单元测试作为评判



SWE-bench 和 SWE-Agent

SWE-bench Verified 衍生版本

- SWE-bench Verified 是由 OpenAI 于 2024 年 8 月发布的评测集
- 包含 500 个由人类专家精标注的问题，是 SWE-bench 的子集+完善版
- 它的难度略微低于 SWE-bench

目前 SWE-bench Verified 已经成为最广泛使用的 Agentic Coding 能力评测集

- 2024 年 8 月，GPT-4o 能够正确解决 33.2%
- 2025 年 1 月，DeepSeek R1 能够正确解决 49.2%
- 2025 年 3 月，Gemini Pro 2.5 能够正确解决 63.8%
- 2025 年 5 月，Claude Sonnet 4 达到 72.7% ~ 80.2%

SWE-bench 和 SWE-Agent

SWE-Agent: 软件工程自主智能体

- 2024 年上半年发布，和 SWE-bench 相同团队，发表于 NeurIPS 2024
- Agentic Coding 领域比较早期的解决方案, 提出了 LLM 与软件开发环境进行交互的抽象
- 基本工作模式遵从 ReAct 模式，即 **推理-行动-观察** 循环

Observation (File Viewer)

```
[File: solvers/diophantine.py (3093 lines total)]
(400 lines above)
401:    elif total_degree == 3:
402:        diop_type = "cubic_thue"
403:
404:    elif (total_degree > 3):
405:        if all(coeff[k] == 1):
406:            diop_type = 'general_sum_even'
407:
408:    if diop_type is not None:
409:        return var, coeff, diop_type
(2684 lines below)
```

(a) Observation from the file viewer.

Response (Thought + Edit Action)

We need to add a check to ensure that all powers are the same. Let's edit the code to include this check.

```
edit 404:407
    elif (total_degree > 3 and
          len(set(k.exp[k.is_Pow])) == 1):
        if all(coeff[k] == 1):
            diop_type = 'general_sum_even'
end_of_edit
```

(b) Action using the edit interface.

SWE-bench 和 SWE-Agent

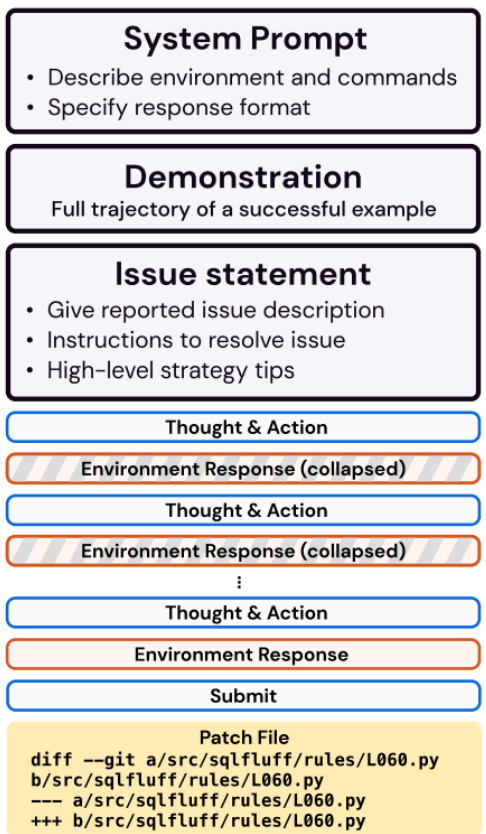
从 LLM 上下文角度，SWE-Agent 工作的“trajectory” 如图所示

这里的 Action-Response 实际上即工具调用

LLM 的 action (tool) 包括：

- find_file
- search_file / search_dir
- open / scroll up / scroll down
- edit (替换文件中的某一个范围的文本)
- Linux 命令行

总结：能找、读、改 源码文件，能用命令行



Anthropic 的软件工程 Agent 方案

虽然 SWE-bench Verified 由 OpenAI 牵头标注，但长期由 Anthropic “霸榜”

2025 年 5 月，最新的 Claude 4 Sonnet 解决该评测中 70% 以上的问题

Anthropic 曾经公布过他们的一些技术方案[1]. 下面简单介绍

[1]https://www.anthropic.com/engineering/swe-bench-sonnet

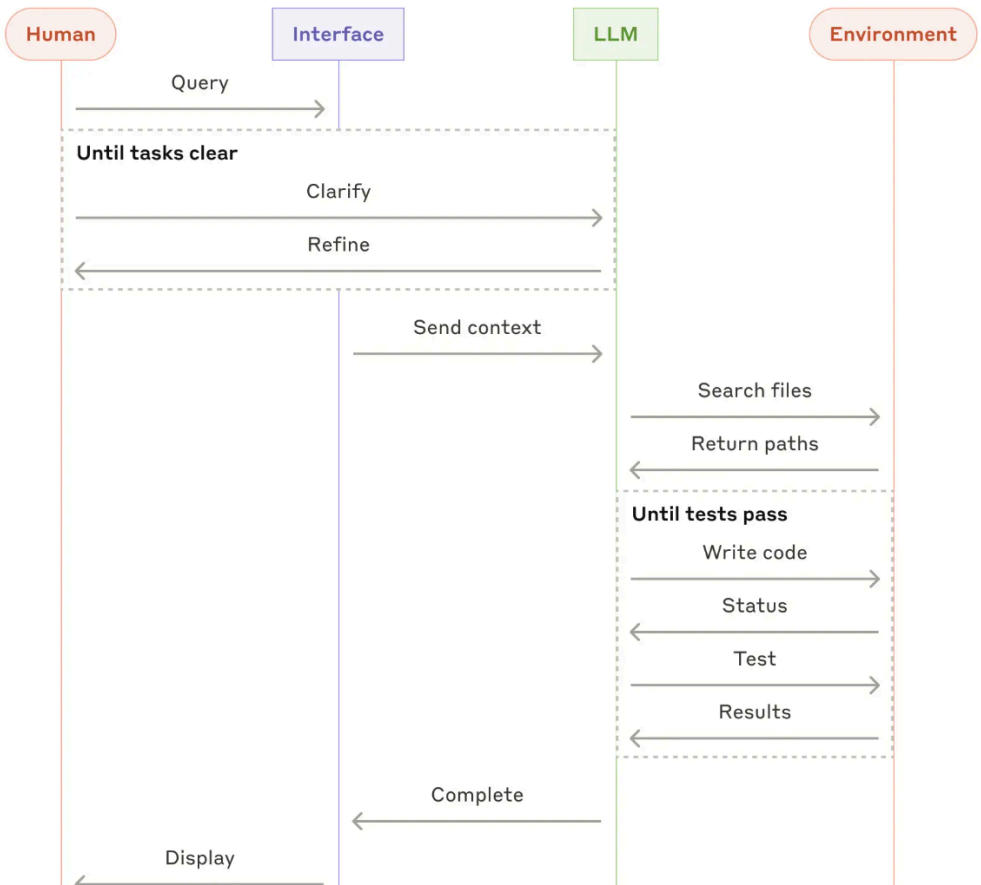
Anthropic 的软件工程 Agent 方案

从实现方式来看，智能体实际上只是 LLM+提示词模板+工具集

- 提示词模板不光负责定义任务、提供问题描述，还要引导 agent 的工作流程
- 工具集提供外部系统交互

从智能体工作流程上看

1. 首先获得任务信息
   - Issue 描述，代码仓库信息等
2. 在代码库中找到与问题相关的源码
3. 编辑源码
4. 执行单元测试，分析测试结果，必要时迭代
5. 任务成功/失败，退出



Anthropic 的软件工程 Agent 方案

他们精心设计了两个 coding agent 工具，从而允许 LLM 在代码环境中完成**几乎任何工作**

Bash Tool

- 在 shell 会话执行一个命令，输入命令字符串，返回命令行输出
- 可以做：配环境、执行脚本、执行单元测试、查看文件等等诸多工作

Editor Tool 相当于一个通过调函数实现的文本编辑器

- 参数 `command` 指定操作类型，包括 **view, create, str_replace, insert, undo_edit**
- 参数 `file_text` 当使用 create 操作时，创建完整文件内容
- 参数 `insert_line` 当使用 insert 命令时，`new_str` 的内容会被插入到 `insert_line` 后
- 参数 `new_str` 新加入的文本内容
- 参数 `old_str` 被替换掉的文本内容
- 参数 `path` 要操作的文件的绝对路径
- 参数 `view_range` 当使用 view 操作时可以指定一个代码行区间

Anthropic 的软件工程 Agent 方案

在以上两个工具和提示词的指导下，Claude 3.5 Sonnet 能够自主尝试完成代码任务

他们展示了一个例子，让 agent 尝试修复某机器学习库里的 bug (任务收录于 SWE-bench Verified)

模型依次做了如下步骤:

1. Editor Tool 创建了一个临时 Python 脚本，尝试复现库中某个 API 的错误
2. Bash Tool 在命令行用 python3 执行脚本，得到了 “TypeError” 错误信息
3. Editor Tool 用 str_replace 操作，修改了一个 method
4. ⋯ 随后的 9 个步骤里模型不断尝试修，最终得到正确的结果

Anthropic 也在报告中承认，该智能体耗时长，费用高。许多成功的 case 耗费了 100k tokens (相当于大约 1 美元)

Outline

总结

Agent 方法的优势:

- 相比于其他 LLM 应用，**外部系统交互**+**自主规划**能突破传统 chatbot 局限
- 上限更高，有潜力解决复杂问题

Agent 方法的潜在问题:

- 需要 LLM 本身有足够强的能力，不稳定，难以复现
- 执行时间难以控制，且成本巨大（调用 Claude 等闭源模型 跑几分钟就是一杯奶茶钱）

对软工任务的影响与启发:

- 在软件工程领域，**代码仓库交互**和**使用命令行工具**能极大提升 AI 能力边界（如 SWE-bench），无限接近人类软件工程师的行为
- 相比于纯语言模型，AI Agent 有自主行动能力，配合上述工具，从而允许自动化解决更复杂的问题