工具与函数调用：机制拆解、混合架构与生态集成

2025 年 10 月 25 日

1 Function Calling 机制（OpenAI, Anthropic）

1.1 流水线概览

在函数调用模式中，LLM 不再直接输出自由文本，而是以结构化参数驱动外部函数。图?? 展示了标准流程：用户请求经过 LLM 解析器生成 JSON 参数，函数路由层按 schema 验证并调用具体工具，最终将结构化结果回传给模型进行回答或直接返回给用户。

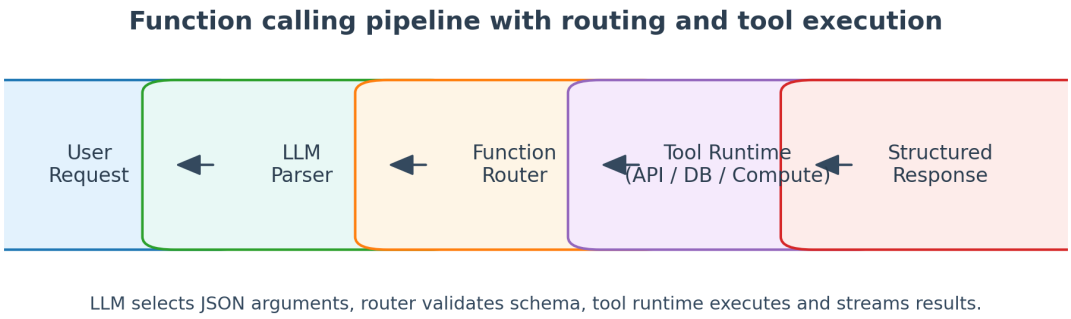

图 1: 函数调用流水线：解析、路由、工具执行与结构化响应。

1.2 OpenAI 与 Anthropic 的实现差异

维度	OpenAI Function Calling	Anthropic Tool Use

接口定义	`functions` 字段，使用 JSON Schema 定义参数；支持多函数候选	`tools` 字段，使用 `input_schema` 指定 JSON Schema；可带 `cache_control`
模型响应	`tool_calls` 数组，包含函数名与参数；可一次返回多个调用	消息流中嵌入 `tool_use` / `tool_result` 片段，保持流式输出
自洽策略	模型可请求 `none` 表示直接回答；客户端决定是否继续循环	Anthropic 建议“tool loop”：模型调用工具—等待结果—继续推理
错误处理	客户端自行捕获异常并回传错误信息供模型重试	可通过 `tool_result` 返回状态码、错误描述；模型可感知失败并修正
安全控制	通过函数白名单、参数校验、代理层保护；支持 Rejection Sampling	支持设置 `max_tool_outputs`，并在 Prompt 中声明安全/合规策略

1.3 函数调用代码示例

Listing 1: OpenAI 函数调用示例：调用天气 API

```python
from openai import OpenAI
import requests

client = OpenAI(api_key="sk-...")

functions = [
    {
        "name": "get_weather",
        "description": "Retrieve weather data for a given city",
        "parameters": {
            "type": "object",
            "properties": {
                "city": {"type": "string"},
                "unit": {"type": "string", "enum": ["celsius", "
                    fahrenheit"]}
            },
            "required": ["city"]
        },
```

```
    }
]

def call_weather(city: str, unit: str = "celsius") -> str:
    resp = requests.get("https://wttr.in", params={"format": "j1", "q":
        city})
    data = resp.json()
    temp = data["current_condition"][0]["temp_C" if unit == "celsius"
       else "temp_F"]
    return f"The temperature in {city} is {temp}°{unit[0].upper()}."

messages = [{"role": "user", "content": "What's the weather like in
   Reykjavik?"}]
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=messages,
    functions=functions,
)

tool_call = response.choices[0].message.tool_calls[0]
args = tool_call.function.arguments
result = call_weather(args["city"], args.get("unit", "celsius"))

messages.extend([
    response.choices[0].message,
    {"role": "tool", "tool_call_id": tool_call.id, "name": "get_weather
       ", "content": result},
])

final = client.chat.completions.create(model="gpt-4o-mini", messages=
   messages)
print(final.choices[0].message.content)
```

2 ReAct + Tool + Memory 混合架构

2.1 框架组合

混合架构将 ReAct（Reason + Act）链式推理、工具调用与记忆系统结合，实现持续对话与上下文感知。流程包括：

- **思考（Thought）**：模型生成下一步策略，决定是否需要工具；

- **行动（Action）**：通过函数调用或插件执行操作；
- **观察（Observation）**：处理返回结果，更新记忆；
- **记忆写入**：将关键信息存入短期对话记忆与长期向量记忆，支撑后续推理。

2.2 记忆设计

- **短期记忆**：存储最近对话与工具交互，便于模型保持语境；
- **长期记忆**：通过向量数据库记录用户偏好、历史任务、重要事实；
- **工作记忆**：任务执行过程中的临时变量、草稿、代码片段，任务完成后可清理或归档。

2.3 安全与治理

- 对记忆写入实施过滤，避免敏感信息泄露；
- 使用 Guardrail 模型审核工具返回结果和模型下一步行动；
- 为不同工具设置频率限制与幂等校验，降低滥用风险。

3 WebAgent / OS-Agent 实现思路

3.1 整体堆栈

WebAgent 与 OS-Agent 通过不同的工具层与上下文缓存对接浏览器或操作系统。图?? 展示了两类 Agent 与共享记忆、治理层的协同。

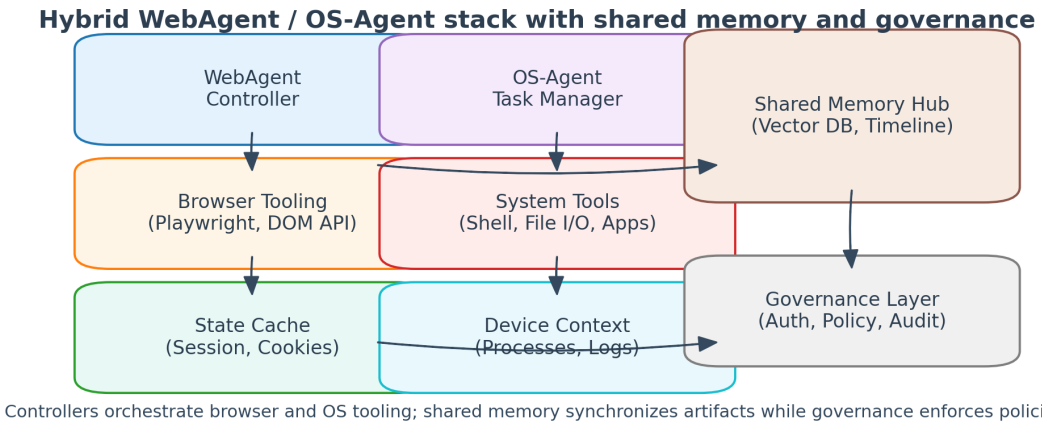

图 2: WebAgent / OS-Agent 混合堆栈：控制器、工具、状态缓存与治理层。

3.2 WebAgent 关键组件

- **浏览器控制**：基于 Playwright、Selenium、puppeteer 等驱动 DOM 交互；
- **页面理解**：结合 HTML 解析、Vision-Language 模型、可视化快照，生成结构化描述；
- **状态管理**：会话、Cookie、localStorage 缓存，支持跨页面任务；
- **安全防护**：防止点击危险链接、执行脚本注入，通过策略引擎进行白名单校验。

3.3 OS-Agent 关键组件

- **系统工具**：终端命令、文件系统、应用程序 API、脚本解释器；
- **设备上下文**：进程列表、日志、权限状态、网络连接；
- **任务管理**：计划任务、进度追踪、回滚机制、异常捕获；
- **治理层**：访问控制、凭证管理、操作审计，确保系统安全。

3.4 共享记忆与协同

WebAgent 与 OS-Agent 可通过共享向量缓存、事件时间线同步任务信息，支持跨平台协作（例如先在浏览器收集资料，再在本地整理报告）。

4 外部 API 与插件系统集成

4.1 插件生态

- **API 插件**：例如 Slack、GitHub、Jira、Salesforce，提供标准化 webhook 与 OAuth 授权；
- **数据插件**：连接数据库、数据湖、知识图谱，提供查询与写入接口；
- **自定义插件**：企业内部服务的封装，需要明确参数 schema、权限策略。

4.2 集成模式

- **直接函数调用**：在函数注册表中定义插件接口，由 LLM 直接选择执行；
- **代理层**：使用中间服务对请求进行二次验证、重试和日志记录；
- **工作流编排**：将插件调用纳入 BPMN / workflow 引擎，与人工审批节点结合。

4.3 API 安全策略

- 使用 JWT/OAuth2 进行身份验证，设置短期令牌并定期轮换；
- 记录调用审计日志，识别异常峰值、重复失败尝试；
- 对响应内容进行过滤与脱敏，避免敏感信息写入记忆。

4.4 插件调用示例

Listing 2: 调用 Notion 数据库插件并写入结果

```python
import requests
import os

NOTION_TOKEN = os.environ["NOTION_TOKEN"]
DATABASE_ID = os.environ["NOTION_DB"]

headers = {
    "Authorization": f"Bearer {NOTION_TOKEN}",
    "Notion-Version": "2022-06-28",
    "Content-Type": "application/json",
}

payload = {
    "parent": {"database_id": DATABASE_ID},
    "properties": {
        "Title": {"title": [{"text": {"content": "Geothermal Report"
            }}]},
        "Status": {"select": {"name": "In Progress"}},
    },
    "children": [
        {"object": "block", "type": "paragraph", "paragraph": {"
            rich_text": [
             {"type": "text", "text": {"content": "Initial findings
                added by tool agent."}}
        ]}}
    ],
}

resp = requests.post("https://api.notion.com/v1/pages", headers=headers
    , json=payload, timeout=15)
resp.raise_for_status()
```

实践建议

- 为函数调用和插件接口统一制定 JSON Schema，便于自动校验与文档化。
- 建立工具调用的回放系统，对异常操作进行可视化排查、回滚与再训练。
- 在混合架构中使用“人类在环”审批节点，确保关键操作受控执行。
- 定期执行安全红队测试，验证工具层和治理层对异常行为的拦截能力。

参考文献

- OpenAI. “Function calling and other API updates.” 2023.
- Anthropic. “Tool Use Capabilities in Claude.” Technical Note, 2024.
- Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models.” ICLR, 2023.
- Qin et al. “WebArena: A Realistic Web Environment for Building Autonomous Agents.” NeurIPS, 2023.
- Microsoft Research. “TaskMatrix.AI: Empowering LLMs with OS-Level Tool Augmentation.” arXiv, 2023.