AI Agent 实战营

开发一个属于你的 **AI Agent**，就从这里开始

Bojie Li

《图解大模型》、《图解 DeepSeek 技术》译者

Pine AI 联合创始人、首席科学家

Press Space for next page →

实战营核心目标

开发一个属于你的 AI Agent，就从这里开始

🎯 掌握核心架构与工程能力

- **深度理解 Agent 架构:** 系统掌握 `LLM + 上下文 + 工具` 的核心设计范式。
- **精通上下文工程:** 掌握从对话历史、用户长期记忆到外部知识库 (RAG) 和文件系统的多层次上下文管理技术。
- **掌握动态工具调用:** 实现 Agent 与外部 API、MCP Server 的可靠集成，并能通过代码生成实现自我进化。
- **构建高级 Agent 模式:** 设计与实现快慢思考 (Mixture-of-Thoughts)、Orchestration 等复杂 Agent 协作模式。

💡 建立系统化开发与部署认知

- **理解技术演进路径:** 洞悉从基础 RAG 到能够自主开发工具的 Agent 的技术演进路径。
- **掌握 Agent 全生命周期:** 具备独立完成 Agent 项目的设计、开发、(使用 LLM as a Judge) 评测与部署的闭环能力。
- **构建领域知识:** 通过法律、学术、编程等多个实战项目，积累跨领域 Agent 开发经验。
- **知识体系沉淀:** 参与共创《深入浅出 AI Agent》书籍，将碎片化知识系统化输出。

9 周实战计划：构建你的通用智能体

周次	主题	内容概览	实战案例
1	**Agent** 入门	Agent 结构与分类、工作流式 vs 自主式	动手搭建一个能联网搜索的 Agent
2	上下文设计	Prompt 模版、对话历史、用户长期记忆	为你的 Agent 增加角色设定和长期记忆
3	**RAG** 与知识库	文档结构化、检索策略与增量更新	构建一个法律知识问答 Agent
4	工具调用与 **MCP**	工具封装与 MCP 接入、外部 API 调用	对接 MCP Server，实现深度调研 Agent
5	编程与代码执行	代码库理解、可靠的代码修改、一致的执行环境	构建一个能自己开发 Agent 的 Agent
6	模型评估与选择	模型能力评估、LLM as a Judge、安全护栏设计	构建评测数据集，用 LLM as a Judge 自动评测 Agent
7	多模态与实时交互	实时语音 Agent、操作电脑与手机	实现语音电话 Agent & 集成 browser-use 操作电脑
8	多 **Agent** 协作	A2A 通信协议、Agent 团队分工与协作	设计多 Agent 协作系统，实现"边打电话边操作电脑"
9	项目集成与展示	Agent 项目总装与展示、最终成果打磨	展示你独一无二的通用 Agent

9 周进阶课题

周次	主题	进阶内容概览	进阶实战案例
1	**Agent** 入门	上下文的重要性	探索上下文缺失对 Agent 行为的影响
2	上下文设计	用户记忆的整理	构建个人知识管理 Agent，实现长文本总结
3	**RAG** 与知识库	长上下文压缩	构建学术论文分析 Agent，总结论文核心贡献
4	工具调用与 **MCP**	从经验中学习	增强深度调研 Agent 的专家能力 (Sub-agent 与领域经验)
5	编程与代码执行	Agent 的自我进化	构建能自主利用开源软件解决未知问题的 Agent
6	模型评估与选择	并行采样与顺序修订	为深度调研 Agent 增加并行与修订能力
7	多模态与实时交互	快慢思考结合	实现快慢思考结合的实时语音 Agent
8	多 **Agent** 协作	Orchestration Agent	用 Orchestration Agent 动态协调电话与电脑操作
9	项目集成与展示	Agent 学习方式对比	对比 Agent 从经验中学习的四种方式

Week 1: Agent 入门

核心内容

Agent 的结构与分类

- 工作流式 **(Workflow-based)**
  - 预定义流程与决策点
  - 确定性高，适合简单业务流程的自动化
- 自主式 **(Autonomous)**
  - 动态规划与自我修正
  - 适应性强，适合开放式研究与探索、解决复杂问题

基础框架与场景判断

- **ReAct** 框架: 观察 → 思考 → 行动
- **Agent = LLM** + 上下文 + 工具
  - LLM: 决策核心 (大脑)
  - 上下文: 感知环境 (眼睛与耳朵)
  - 工具: 与世界交互 (双手)

Week 1 · 实战案例

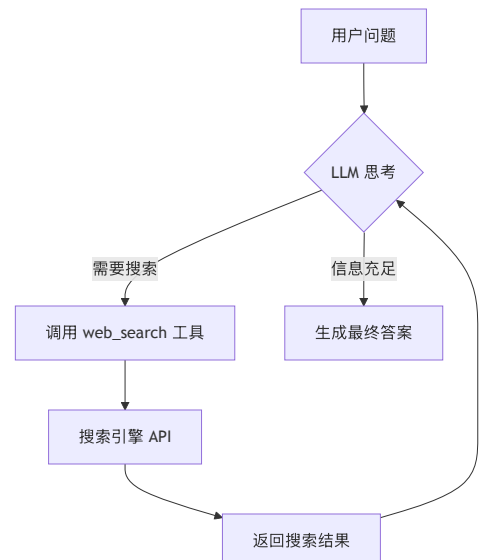
动手搭建一个能联网搜索的 Agent

目标: 构建一个基础的自主式 Agent，能够理解用户问题，通过搜索引擎获取信息，并总结出答案。

核心挑战

- **任务分解:** 将复杂问题分解为可搜索的关键词。
- **工具定义:** 定义并实现一个 `web_search` 工具。
- **结果整合:** 理解搜索结果，并综合信息生成最终答案。

架构设计



Week 1 · 进阶内容

上下文的重要性：Agent 的操作系统

核心理念: `The context is the agent's operating system.` 上下文是 Agent 感知世界、做出决策、记录历史的唯一依据。

思考 (Thinking)

- Agent 的内心独白和推理链。
- **缺失后果:** 导致 Agent 行为黑盒，无法调试和理解其决策过程。

工具调用 (Tool Call)

- Agent 决定采取的行动，记录其意图。
- **缺失后果:** 无法追踪 Agent 的行为历史，难以复盘。

工具结果 (Tool Result)

- 行动带来的环境反馈。
- **缺失后果:** Agent 无法感知其行为的后果，可能导致无限重试或错误规划。

Week 1 · 进阶实践

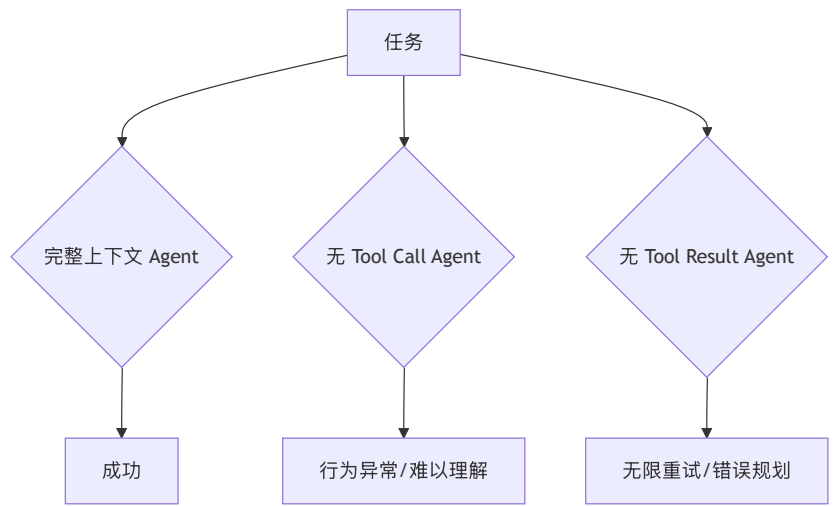
探索上下文缺失对 Agent 行为的影响

目标: 通过实验，理解 `thinking`，`tool call`，`tool result` 各部分在 Agent 工作流中不可或缺的作用。

核心挑战

- **修改 Agent 框架**: 修改 Agent 的核心循环，选择性地从上下文中移除特定部分。
- **设计对比实验**: 设计一组任务，在这些任务中，缺失不同上下文的 Agent 会表现出明显的行为差异甚至失败。
- **行为分析**: 分析并总结不同上下文缺失分别导致了哪种类型的失败。

实验设计



Week 2: 上下文设计 (Context Engineering)

核心内容

Prompt 模版

- **系统提示词:** 设定 Agent 的角色、能力边界和行为准则。
- **工具集:** 工具的名字、说明、参数。

对话历史与用户记忆

- **事件序列:** 将对话历史建模为 "观察" 与 "行动" 的交替序列。
- **用户长期记忆:** 将对话中用户的关键信息（如偏好、个人信息）提取并结构化保存，用于未来的交互。

Week 2 · 实战案例

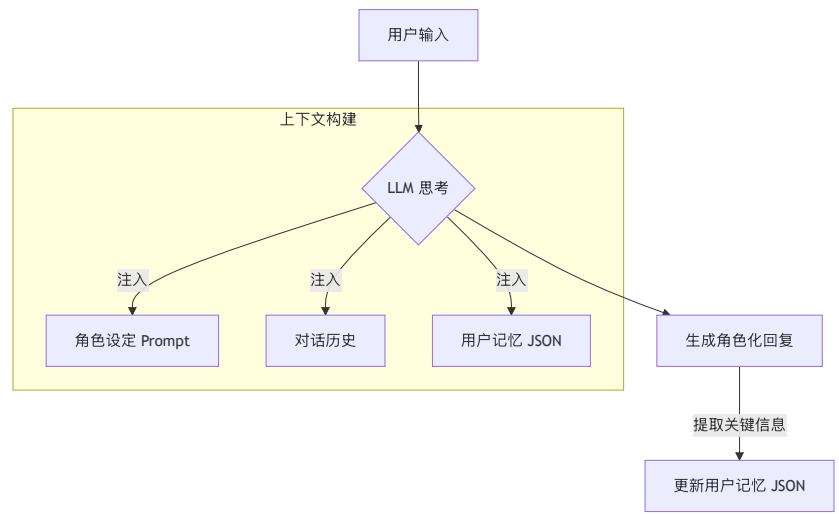
为你的 Agent 增加角色设定和长期记忆

目标: 提升 Agent 的个性化与连续服务能力。Agent 需要能模仿特定角色（如动漫人物）的风格说话，并能记住用户的关键信息（如姓名、兴趣），在后续对话中应用这些记忆。

核心挑战

- **角色扮演:** 如何在 Prompt 中清晰地定义角色的语言风格和个性，并让 Agent 稳定地保持人设。
- **记忆提取与存储:** 如何从非结构化的对话中，准确提取关键信息并存为一个结构化的 JSON 对象。
- **记忆应用:** 如何将存储的用户记忆 JSON 自然地融入到后续对话的 Prompt 中，让 Agent 看起来真的"记住"了用户。

架构设计



Week 2 · 进阶内容

用户记忆的整理

核心理念: 简单的记忆拼接会导致上下文膨胀、信息冲突和过时。高级的记忆系统需要在后台对用户的长期记忆进行持续的整理、去重、修正和总结，形成一个动态演进的用户画像。

实现策略

- **记忆去重与合并:** 识别并合并内容相似或重复的记忆条目。
- **冲突解决:** 当新的记忆与旧的记忆发生冲突时（如用户更改了偏好），以最新的信息为准。
- **定期总结:** 定期或在后台空闲时，使用 LLM 对零散的记忆点进行总结，提炼出更高层次的用户偏好和特征。

架构设计

Week 2 · 进阶实践

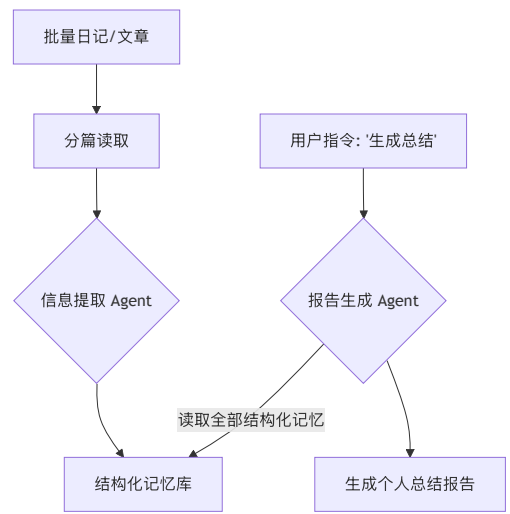
将你的日记总结成个人报告

目标: 构建一个能够处理大量个人文本（如每日日记、博客文章）的 Agent，通过对这些文本的阅读和整理，最终生成一篇详尽、清晰的个人总结报告。

核心挑战

- **长文本处理:** 如何处理总量可能超过 LLM 上下文窗口的日记/文章。
- **信息提炼与结构化:** 如何从叙事性的文本中，提取出结构化的信息点（如关键事件、情绪变化、个人成长）。
- **连贯的总结生成:** 如何将零散的信息点，组织成一篇逻辑连贯、可读性强的总结报告。

架构设计



Week 3: RAG 系统与知识库

核心内容

文档结构化与检索策略

- **Chunking**: 将长文档切分为有意义的语义块。
- **Embedding**: 将文本块向量化，用于相似度检索。
- **混合检索**: 结合向量相似度与关键词检索，提高召回率与精确度。
- **重排序 (Re-ranking)**: 使用更复杂的模型对初步检索结果进行二次排序。

基础 RAG

- **知识表达**: 使用清晰、结构化的自然语言表达知识。
- **知识库构建**: 将文档处理并载入向量数据库。
- **精准检索**: 根据用户问题，精准定位知识库中的相关条目。

Week 3 · 实战案例

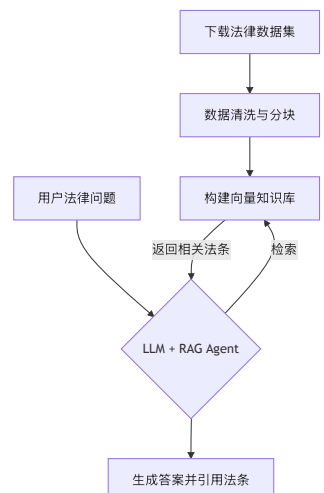
构建一个法律知识问答 Agent

目标: 让 Agent 成为一个专业的法律顾问。我们将使用公开的中国刑法/民法数据集构建一个知识库，使 Agent 能够准确回答用户的法律问题，并明确指出答案所依据的具体法条。

核心挑战

- **领域数据处理:** 如何解析和清洗结构化的法律条文数据，并优化其在 RAG 系统中的检索效果。
- **答案的精确性与溯源:** Agent 的回答必须严格基于知识库内容，避免自由发挥，并且必须提供法条来源。
- **处理模糊查询:** 如何引导用户提出更明确的问题，以匹配到最相关的法律条文。

架构设计



Week 3 · 进阶内容

将文件系统作为终极上下文

核心理念: `Treat the file system as the ultimate context.` Agent 不应将巨大的观测结果（如网页、文件内容）直接塞入上下文，这会导致成本高昂、性能下降且有窗口限制。正确的做法是，将这些大数据存入文件，只在上下文中保留一个轻量的"指针"（摘要和文件路径）。

实现策略

- **可恢复压缩:** 当工具返回大量内容时（如 `read_file`），先将其完整保存到沙箱的文件系统中。
- **摘要与指针:** 只将内容的摘要和文件路径追加到主上下文中。
- **按需读写:** Agent 通过 `read_file` 工具，可以在后续步骤中按需从文件系统读取完整内容。

架构设计

Week 3 · 进阶实践

构建一个能阅读多篇论文的 Agent

目标: 训练一个学术研究 Agent，它能够阅读一篇指定的论文及其所有的参考文献（通常是几十篇 PDF），并在此基础上，总结出该论文相比于其参考文献的核心贡献与创新点。

核心挑战

- **海量 PDF 处理:** 如何高效地解析数十篇 PDF 论文，并提取关键信息（摘要、结论、方法论）。
- **跨文档关联分析:** 核心挑战在于，Agent 需要在主论文和多篇参考文献之间建立关联，进行比较分析，而不是简单地总结单篇论文。
- **贡献点提炼:** 如何从复杂的学术论述中，精准地提炼出论文的"增量贡献"。

架构设计

Week 4: 工具调用与 MCP

核心内容

多种工具封装方式

- **函数调用 (Function Calling)**: 将本地代码函数直接暴露给 Agent。
- **API 接入**: 调用外部 HTTP API，获取实时数据或执行远程操作。
- **Agent as a Tool**: 将一个专有 Agent (如代码生成 Agent) 封装为另一个 Agent 可调用的工具。

MCP (Model Context Protocol)

- **标准化接口**: 为模型与外部工具/数据源提供一个统一、语言无关的连接标准。
- **即插即用**: 开发者可以发布符合 MCP 规范的工具，Agent 可以动态发现并使用它们。
- **安全与隔离**: 内置权限和沙箱机制，确保工具调用的安全性。

Week 4 · 实战案例

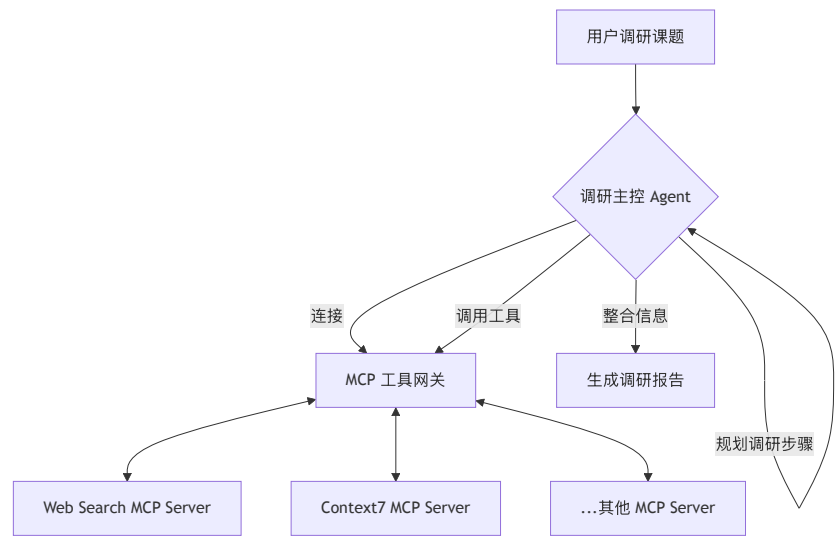
对接 MCP Server，实现深度调研 Agent

目标: 构建一个能进行深度信息调研的 Agent。它需要能连接到多个符合 MCP 规范的外部工具服务器，并能自主规划、调用这些工具来完成一个复杂的调研课题。

核心挑战

- **权威信息源识别:** Agent 需要在海量信息中，精准识别并采纳官方文档、学术论文等高可信度的信息源。
- **多工具协同:** 如何规划一个调用链，让多个工具（如先搜索、再读取、再分析）的输出/输入串联起来，形成完整的工作流。
- **开放式问题探索:** 如何处理没有唯一答案的开放式问题，进行多角度的探索性搜索并汇总结果。

架构设计



Week 4 · 进阶内容

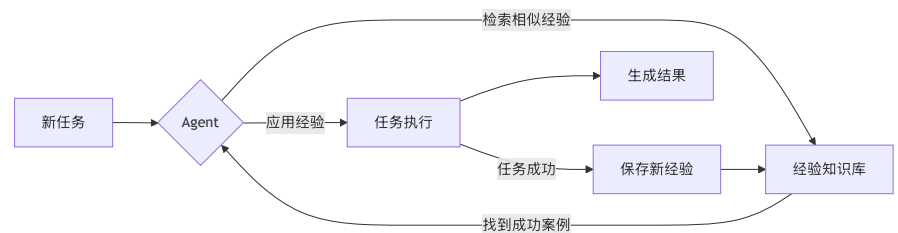
从经验中学习：越做越熟练

核心理念: 真正的智能体不仅要会使用工具，更要能从使用工具的经验中学习和进化。它应该能记住成功解决某类任务的"套路"（即 Prompt 模板和工具调用序列），并在未来遇到相似任务时直接复用。

实现策略

- **经验存储:** 当一个复杂任务成功完成后，Agent 会将整个过程（包括用户意图、思考链、工具调用序列、最终结果）作为一个"经验案例"存入知识库。
- **经验检索:** 面对新任务时，Agent 首先在经验库中搜索相似案例。
- **经验应用:** 如果找到相似案例，Agent 会将该案例的成功策略作为高级指导，而不是每次都从零开始思考。

架构设计



Week 4 · 进阶实践

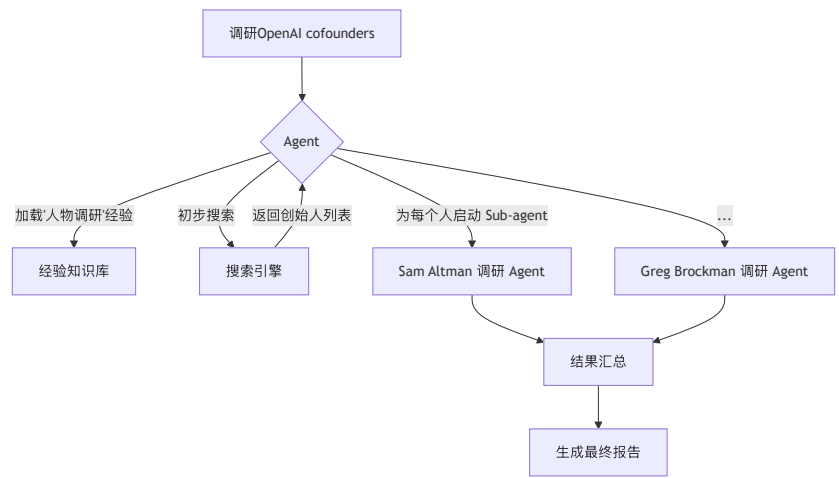
增强深度调研 Agent 的专家能力

目标: 针对深度调研中的复杂场景，为 Agent 赋予专家级的处理能力。例如，在调研"OpenAI 的联合创始人"时，能自动为每一位创始人启动一个并行的子调研 Agent；在搜索人物信息时，能有效处理重名问题。

核心挑战

- **加载领域经验**: 如何根据任务类型（如"学术调研" vs "人物调研"），加载不同的经验知识，指导 Agent 使用最合适的权威信息源和 Prompt 策略。
- **动态 Sub-agent**: 如何让主 Agent 根据初步搜索结果，动态地创建多个并行的子 Agent 来分别处理子任务。
- **歧义消解**: 在处理人物搜索等易产生歧义的场景时，如何设计澄清和验证机制。

架构设计



Week 5: 编程与代码执行

代码 Agent 的核心挑战

- **代码库理解:**
  - 如何从大代码库中查找相关代码（语义搜索）?
  - 如何准确查询代码中函数的所有引用点?
- **可靠的代码修改:**
  - 如何可靠地将 AI 生成的 diff 应用到源文件 (`old_string` -> `new_string`)?
- **一致的执行环境:**
  - 如何保证 Agent 每次执行命令都在同一个终端会话中 (继承 `pwd` , `env var` 等)?
  - 如何为 Agent 的执行环境预先配置好所需的依赖和工具?

Week 5 · 实战案例

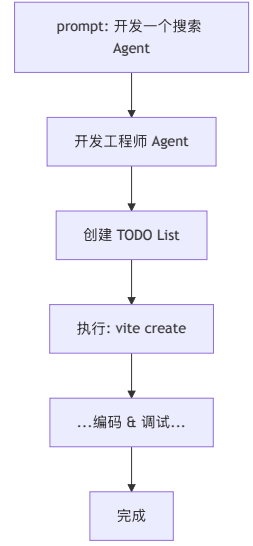
构建一个能自己开发 Agent 的 Agent

目标: 打造一个 "Agent 开发工程师" Agent。它能接收一个高层级的自然语言需求（例如："开发一个能上网搜索的 Agent，前端使用 React + Vite + Shadcn UI，后端使用 FastAPI..."），然后自主完成整个应用的开发。

核心挑战

- **文档驱动开发:** 如何让 Agent 先为要开发的应用撰写设计文档，并严格遵循该文档进行后续代码实现。
- **测试驱动开发:** 如何确保 Agent 为其生成的每一段代码都编写并执行测试用例，保证最终交付应用的质量和正确性。
- **开发和测试环境:** Agent 需要有良好的开发和测试环境，才能自主执行测试用例，发现 bug，进而修复 bug。

架构设计



Week 5 · 进阶内容

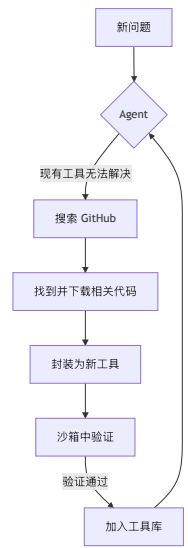
Agent 的自我进化：自主开发工具

核心理念: Agent 能力的终极形态是自我进化。当面对一个现有工具无法解决的问题时，一个高级 Agent 不应该放弃，而应该利用其代码编写能力，为自己创造一个新工具。

实现策略

- **能力边界识别:** Agent 首先需要判断当前问题是否超出了其现有工具集的能力范围。
- **工具创造规划:** Agent 规划出新工具的功能、输入、输出，并搜索开源代码库（如 GitHub）寻找可用的实现。
- **代码封装与验证:** Agent 将找到的代码封装成一个新的工具函数，并为其编写测试用例，在沙箱中验证其正确性。

架构设计



Week 6: 大模型的评估和选择

核心内容

评估大模型的能力边界

- **核心能力维度:** 智力、知识量、幻觉、长文本、指令遵循、工具调用。
- **构建有区分度的测试用例:** 设计 Agent-centric 的评测集，而非简单的 Chatbot 问答。
- **LLM as a Judge**: 使用一个强大的 LLM (如 GPT-4.1) 作为"裁判"，来自动化地评估和比较不同模型或 Agent 的输出质量。

为大模型装上安全护栏

- **输入过滤:** 防止恶意提示词注入。
- **输出过滤:** 监测并拦截不当或危险的输出内容。
- **人工介入:** 在高风险操作前，引入人工确认环节 (Human-in-the-loop)。
- **成本控制:** 监控 token 消耗，设置预算限制，防止滥用。

Week 6 · 实战案例

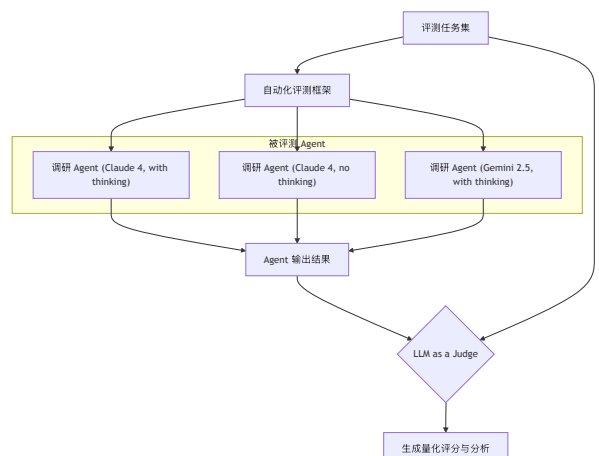
构建评测数据集，用 LLM as a Judge 自动评测 Agent

目标: 为我们前几周构建的深度调研 Agent，系统性地构建一个评测数据集。然后，开发一个自动化的测试框架，使用 LLM as a Judge 的方法，评测不同"大脑"（如 Claude 4 vs Gemini 2.5）以及不同策略（如打开/关闭思考链）对 Agent 性能的影响。

核心挑战

- **评测数据集设计:** 如何设计一组既有代表性又能覆盖各种边界情况的调研任务？
- **"裁判" Prompt 设计:** 如何设计给 "LLM Judge" 的 Prompt，才能让它公平、一致、准确地对 Agent 的输出进行打分？
- **结果的可解释性:** 如何从自动评测的结果中，分析出不同模型或策略的优劣势所在。

架构设计



Week 6 · 进阶内容

并行采样与顺序修订

核心理念: 模拟人类的"集思广益"与"反思修正"过程，以应对复杂和开放性问题，提升 Agent 输出的质量和鲁棒性。

并行采样 (Parallel Sampling)

- **思路:** 同时启动多个 Agent 实例，使用略微不同的 Prompt 或更高的 temperature，从多个角度并行探索解决方案。
- **优势:** 增加找到最优解的概率，避免单一 Agent 的思维局限。
- **实现:** 类似 Multi-Agent，但目标是解决同一问题，最后通过评估机制（如 LLM as a Judge）筛选出最佳答案。

顺序修订 (Sequential Revision)

- **思路:** 让 Agent 对自己的初步输出进行自我批判和修正。
- **流程:** 初始响应 → 自我评估 → 识别问题 → 生成改进 → 最终输出。
- **优势:** 提升单次任务的成功率和答案的深度，实现自我优化。

Week 6 · 进阶实践

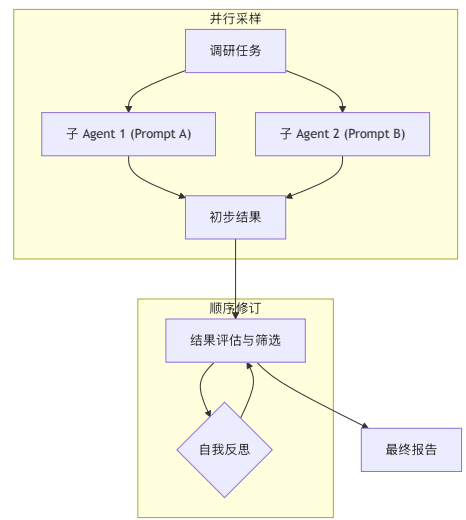
为深度调研 Agent 增加并行与修订能力

目标: 将并行采样和顺序修订两种高级策略，集成到我们的深度调研 Agent 中。并通过我们刚刚构建的评测框架，量化评估这两种策略是否以及在多大程度上提升了 Agent 的性能。

核心挑战

- **策略融合:** 如何将并行采样（横向扩展）和顺序修订（纵向深化）有机地结合到一个 Agent 工作流中？
- **成本控制:** 这两种策略都会显著增加 LLM 的调用成本，如何设计机制在性能提升和成本之间取得平衡？
- **性能归因:** 如何在评测中，准确地将性能提升归因于并行采样还是顺序修订？

架构设计



Week 7: 多模态与实时交互

核心内容

实时语音电话 Agent

- **技术栈:** VAD (语音活动检测), ASR (语音识别), LLM, TTS (语音合成)。
- **低延迟交互:** 优化从用户语音输入到 Agent 语音输出的端到端延迟。
- **自然的打断处理:** 允许用户在 Agent 讲话时随时插入，实现更接近人类的对话流。

操作电脑和手机

- **视觉理解:** Agent 需要理解屏幕截图，识别 UI 元素 (按钮、输入框、链接)。
- **操作映射:** 将 "点击登录按钮" 这样的自然语言指令，精确映射到屏幕坐标或 UI 元素ID。
- **现有框架集成:** 直接调用 `browser-use` 等成熟框架，快速赋予 Agent 操作电脑的能力。

Week 7 · 实战案例 (1/2)

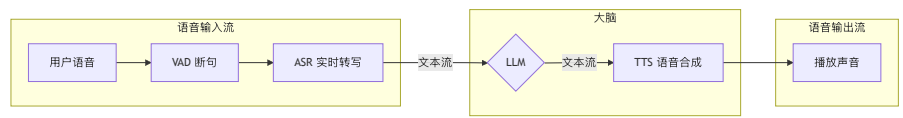
实现能听会说的实时语音电话 Agent

目标: 从零开始，自己动手构建一个能够与用户进行实时、流畅语音对话的 Agent。它需要能够快速响应，理解并执行语音指令，甚至能主动发起引导式对话。

核心挑战

- **延迟控制:** 从用户语音输入到 Agent 语音输出的端到端延迟，是决定体验好坏的关键。如何优化技术栈的每个环节?

架构设计



Week 7 · 实战案例 (2/2)

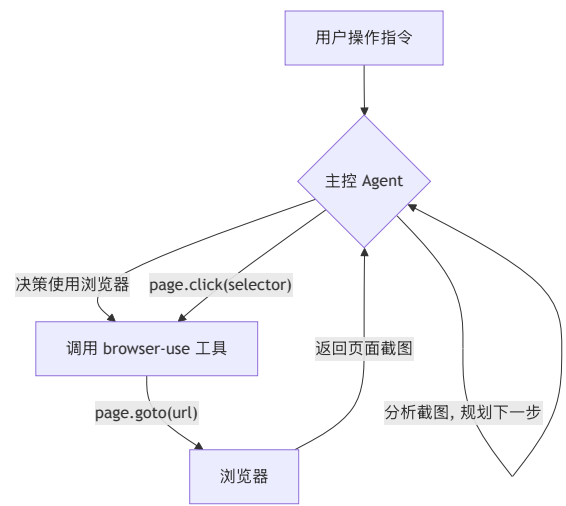
集成 browser-use，让 Agent 操作你的电脑

目标: 调用现有的 `browser-use` 框架，让我们的 Agent 具备操作电脑浏览器的能力。Agent 需要能理解用户的操作指令（如"帮我打开 Anthropic 官网，找到 computer use 的文档"），并将其转化为对浏览器的实际操作。

核心挑战

- **框架集成:** 如何将 `browser-use` 作为一个工具，平滑地集成到我们现有的 Agent 架构中。
- **指令泛化:** 用户指令可能是模糊的，如何让 Agent 理解这些指令并转化为 `browser-use` 支持的精确操作。
- **状态同步:** 如何让 Agent 感知到浏览器操作的结果（如页面跳转、元素加载），以进行下一步的决策。

架构设计



Week 7 · 进阶内容

快慢思考与智能交互管理

快慢思考 (Mixture-of-Thoughts) 架构

- **快速响应路径**: 利用低延迟模型 (如 Gemini 2.5 Flash) 实现即时反馈，处理简单查询和维持对话流畅性。
- **深度思考路径**: 利用能力更强的 SOTA 模型 (如 Claude 4 Sonnet) 进行复杂推理和工具调用，为用户提供更精准、深入的回答。

智能交互管理

- **聪明的打断 (Interrupt Intent Detection)**: 通过 VAD 和小模型过滤背景噪声和无意义的附和，只在用户有明确打断意图时才中止发言
- **发言权判断 (Turn Detection)**: 分析用户已说出内容的语义完整性，判断 AI 是否应该继续发言，避免抢话
- **沉默管理 (Silence Management)**: 在用户长时间沉默时，主动开启新话题或进行追问，保持对话的连贯性

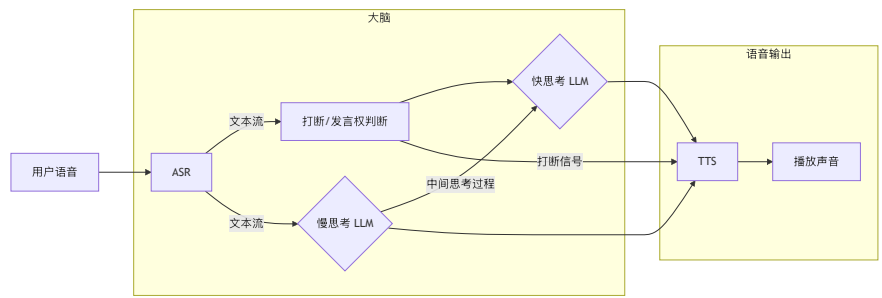
Week 7 · 进阶实践

实现高级实时语音 Agent

目标: 构建一个集成了"快慢思考"架构与"智能对话管理"的高级语音 Agent，使其在响应速度和交互自然度上都达到业界领先水平。

- **基础推理:** 提问："8 的 6 次方等于多少？"——需在 2 秒内做出初步回应，15 秒内给出正确答案 "262144"。
- **工具调用:** 提问："北京今天天气如何？"——需在 2 秒内回应，15 秒内通过 API 返回准确天气。
- **智能交互管理:**
  - **智能打断:** 在 Agent 发言过程中：
    - 用户说“嗯”，Agent 不应停止说话。
    - 用户拍一下桌子，Agent 不应停止说话。
    - 用户说 "那它的续航..." 时，Agent 应立即中止当前发言。
  - **发言权判断:** 用户说 "那它的续航..." 后故意停顿，Agent 不应回应。
  - **沉默管理:** 用户说 "那它的续航..." 后停顿超过 3 秒，Agent 能主动引导对话或追问，保持交流流畅。

架构设计



Week 8: 多 Agent 协作

核心内容

单 Agent 的局限

- **上下文成本高昂:** 单一上下文窗口在复杂任务中迅速膨胀。
- **顺序执行效率低下:** 无法并行处理多个子任务。
- **长上下文质量下降:** 模型在过长的上下文中容易"遗忘"或"分心"。
- **无法并行探索:** 只能沿着单一路径进行探索。

Multi-Agent 的优势

- **并行处理:** 将任务分解，交给不同 SubAgent 并行处理，提升效率。
- **独立上下文:** 每个 SubAgent 拥有独立的、更专注的上下文窗口，保证执行质量。
- **压缩即本质:** 每个 SubAgent 只需返回其最重要的发现，由主 Agent 聚合，实现高效的信息压缩。
- **集体智能涌现:** 适合开放式研究等需要多角度分析的任务。

Week 8 · 实战案例

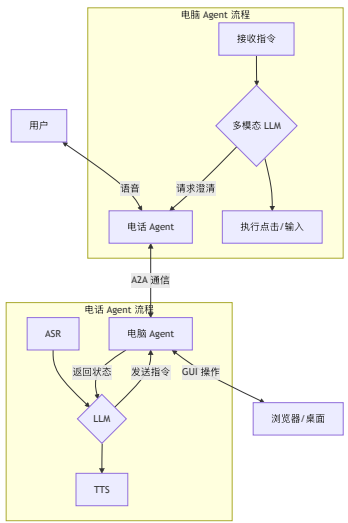
设计一个多 Agent 协作系统, 实现"边打电话边操作电脑"

目标: 解决"一心二用"的难题。构建一个由"电话 Agent"和"电脑 Agent"组成的团队。"电话 Agent" 负责与用户语音沟通, 获取信息; "电脑 Agent" 负责同步操作网页。两者实时通信, 高效协同。

核心挑战

- **双 Agent 架构**: 两个独立的 Agent，一个负责语音通话 (电话 Agent)，一个负责操作浏览器 (电脑 Agent)。
- **Agent 间协同通信**: 两个 Agent 必须能高效双向通信。电话 Agent 获取的信息需立刻告知电脑 Agent，反之亦然。这可以通过工具调用实现。
- **并行工作与实时性**: 关键在于两个 Agent 必须能并行工作，互不阻塞。各自的上下文中，都需要包含来自对方 Agent 的实时消息。

架构设计



Week 8 · 进阶内容

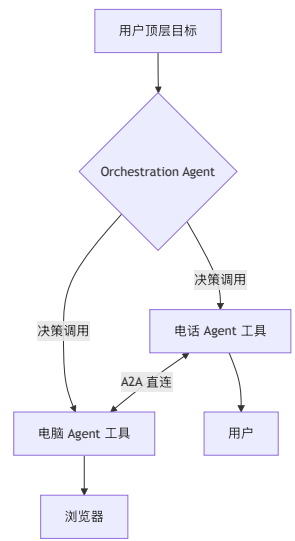
Orchestration Agent: 将 Sub-agent 作为工具

核心理念: 不再是硬编码的 Agent 间协作, 而是引入一个更高层级的 "Orchestration Agent"。它的核心职责是理解用户顶层目标, 并动态地选择、启动和协调一组"专家 Sub-agent" (作为工具) 来共同完成任务。

实现策略

- **Sub-agent as Tools**: 每个专家 Sub-agent (如电话 Agent, 电脑 Agent, 调研 Agent) 都被封装成一个符合标准接口的"工具"。
- **动态工具调用**: Orchestration Agent 根据用户需求, 异步地调用一个或多个 Sub-agent 工具。
- **Agent 间直接通信**: 允许被调用的 Sub-agent 之间建立直接的通信渠道, 用于高效的任务协同, 而无需事事通过 Orchestration Agent 中转。

架构设计



Week 8 · 进阶实践

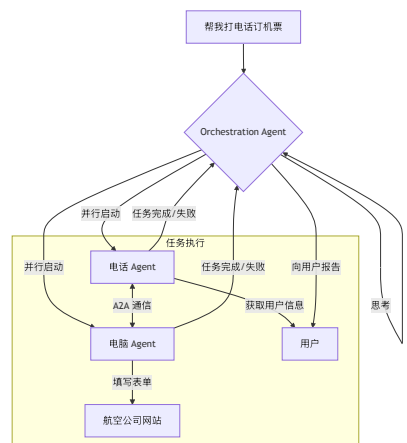
用 Orchestration Agent 动态协调电话与电脑操作

目标: 重构我们"边打电话边操作电脑"的系统。不再硬编码启动两个 Agent, 而是创建一个 Orchestration Agent。当用户提出"帮我打电话预定一个航班"的需求时, Orchestration Agent 能自动理解这个任务需要"打电话"和"操作电脑"两种能力, 于是并行地启动这两个 Sub-agent, 并让它们协同工作。

核心挑战

- **任务规划与工具选择**: Orchestration Agent 如何准确地将一个模糊的用户目标, 分解为需要哪些具体的 Sub-agent 工具。
- **异步工具管理**: 如何管理多个并行执行、长时间运行的 Sub-agent 工具的生命周期 (启动、监控、终止)。
- **Sub-agent 间通信**: 如何为动态启动的 Sub-agent 建立一个高效、临时的直接通信机制。

架构设计



Week 9: 项目展示

核心内容

项目总装与展示

- **整合能力:** 将前 8 周学习到的各项能力 (RAG, 工具调用, 语音, 多模态, 多 Agent) 整合到一个最终项目中。
- **成果展示:** 每位学员将有机会展示自己独一-无二的通用 Agent，分享创作过程中的思考与挑战。
- **同行评审:** 通过互相演示和提问，从其他同学的项目中获得启发和灵感。

图书打磨与总结

- **知识沉淀:** 共同回顾和总结 9 周的核心知识点，将其固化为最终的《深入浅出 AI Agent》书稿。
- **内容共创:** 对书稿内容提出修改建议，共同打磨，确保其"系统实用"。
- **署名出版:** 所有参与共创的学员，名字都将出现在最终出版的实体书上。

Week 9 · 实战案例

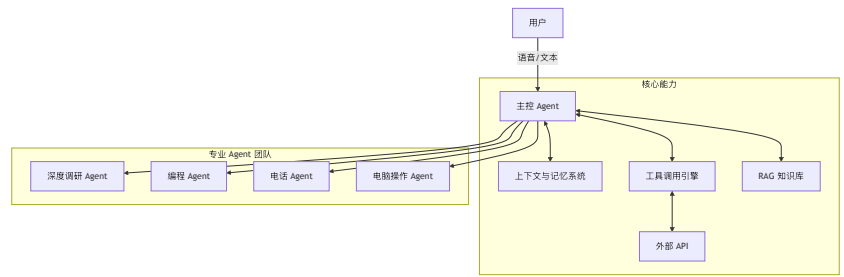
展示你独一无二的通用 Agent

目标: 对训练营期间构建的个人 Agent 项目进行一次全面的总结和展示。这不仅是一次成果汇报，更是一次将所学知识体系化、向他人清晰阐述复杂技术方案的综合能力演练。

展示要点

- **Agent 定位:** 你的 Agent 解决了什么核心问题?
- **技术架构:** 你是如何综合运用所学知识 (上下文, RAG, 工具, 多模态, 多 Agent) 来实现目标的?
- **创新亮点:** 你的 Agent 最具创意的设计是什么?
- **Demo 演示:** 现场演示 Agent 的核心功能。
- **未来展望:** 你计划如何继续迭代和完善你的 Agent?

最终项目架构示例



Week 9 · 进阶内容 (1/2)

Agent 从经验中学习的四种方式

1. 依赖长上下文能力

- **思路:** 相信并利用模型自身的长上下文处理能力，将完整的、未经压缩的对话历史作为输入。
- **实现:**
  - **保留最近对话:** 完整保留最近的交互历史 (Context Window)。
  - **压缩长时记忆:** 利用 `Linear Attention` 等技术，将遥远的对话历史自动压缩到 Latent Space 中。
  - **提取关键片段:** 利用 `Sparse Attention` 等技术，让模型从遥远的对话历史中自动提取与当前任务最相关的片段。
- **优点:** 实现最简单，能最大程度保留原始信息细节。
- **缺点:** 对模型能力依赖强。

2. 文本形式提取 (RAG)

- **思路:** 将经验总结成自然语言，存入知识库。
- **实现:** 通过 RAG 检索相关的经验文本并注入 Prompt。
- **优点:** 成本可控，知识可读可维护。
- **缺点:** 依赖检索的准确性。

Week 9 · 进阶内容 (2/2)

Agent 从经验中学习的四种方式

3. 后训练 (SFT/RL)

- **思路:** 将经验学进模型权重。
- **实现:** 将高质量的 Agent 行为轨迹作为数据，对模型进行微调 (SFT) 或强化学习 (RL)。
- **优点:** 将经验内化为模型的"直觉"，适合复杂任务，泛化能力强。
- **缺点:** 成本较高，需要大量高质量数据；周期较长，很难实现实时的经验反馈循环，即线上刚刚失败的例子马上不会犯类似错误。

4. 抽象为代码 (工具/Sub-agent)

- **思路:** 将重复出现的成功模式，抽象成一个可复用的工具或 Sub-agent。
- **实现:** Agent 识别出可自动化的模式，并编写代码将其固化。
- **优点:** 可靠、高效的学习方式。
- **缺点:** 对 Agent 的代码能力要求高；工具数量较大后，工具选择成为挑战。

Week 9 · 进阶实践

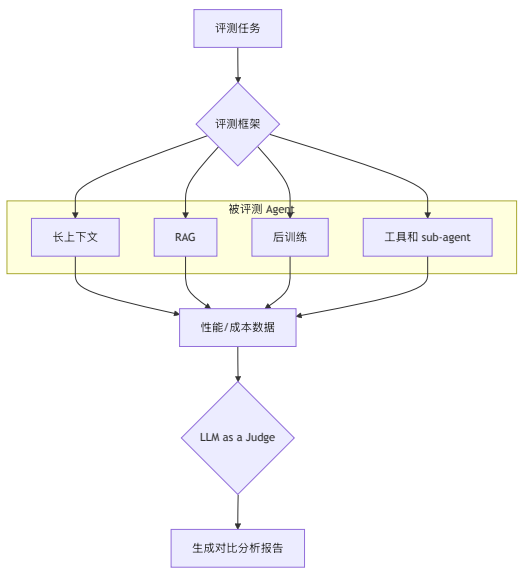
对比 Agent 从经验中学习的四种方式

目标: 使用我们在第 6 周构建的评测框架，设计实验，来对比 Agent 从经验中学习的四种方式的优缺点。

核心挑战

- **实验设计**: 如何设计一组任务，能够清晰地体现出四种不同学习方式的差异?
- **成本与性能权衡**: 如何在评测报告中，将每种方法的"性能得分"与其"计算成本"相结合，进行综合评估?
- **场景化分析**: 得出结论，在什么样的任务场景下，应该优先选择哪种学习方式?

架构设计



总结回顾

9 周实战计划：构建你的通用智能体

周次	主题	内容概览	实战案例
1	**Agent** 入门	Agent 结构与分类、工作流式 vs 自主式	动手搭建一个能联网搜索的 Agent
2	上下文设计	Prompt 模版、对话历史、用户长期记忆	为你的 Agent 增加角色设定和长期记忆
3	**RAG** 与知识库	文档结构化、检索策略与增量更新	构建一个法律知识问答 Agent
4	工具调用与 **MCP**	工具封装与 MCP 接入、外部 API 调用	对接 MCP Server，实现深度调研 Agent
5	编程与代码执行	代码库理解、可靠的代码修改、一致的执行环境	构建一个能自己开发 Agent 的 Agent
6	模型评估与选择	模型能力评估、LLM as a Judge、安全护栏设计	构建评测数据集，用 LLM as a Judge 自动评测 Agent
7	多模态与实时交互	实时语音 Agent、操作电脑与手机	实现语音电话 Agent & 集成 browser-use 操作电脑
8	多 **Agent** 协作	A2A 通信协议、Agent 团队分工与协作	设计多 Agent 协作系统，实现"边打电话边操作电脑"
9	项目集成与展示	Agent 项目总装与展示、最终成果打磨	展示你独一无二的通用 Agent

9 周进阶课题

周次	主题	进阶内容概览	进阶实战案例
1	**Agent** 入门	上下文的重要性	探索上下文缺失对 Agent 行为的影响
2	上下文设计	用户记忆的整理	构建个人知识管理 Agent，实现长文本总结
3	**RAG** 与知识库	长上下文压缩	构建学术论文分析 Agent，总结论文核心贡献
4	工具调用与 **MCP**	从经验中学习	增强深度调研 Agent 的专家能力 (Sub-agent 与领域经验)
5	编程与代码执行	Agent 的自我进化	构建能自主利用开源软件解决未知问题的 Agent
6	模型评估与选择	并行采样与顺序修订	为深度调研 Agent 增加并行与修订能力
7	多模态与实时交互	快慢思考结合	实现快慢思考结合的实时语音 Agent
8	多 **Agent** 协作	Orchestration Agent	用 Orchestration Agent 动态协调电话与电脑操作
9	项目集成与展示	Agent 学习方式对比	对比 Agent 从经验中学习的四种方式

欢迎加入 AI Agent 实战营

开发一个属于你的 AI Agent，就从这里开始

Powered by Slidev