Module 2 -- Experiment Agent Tutorial

背景介绍

动机

针对已有的研究任务，deepresearch检索到若干相关的SOTA工作。目前科研的流程就是人工将这若干的SOTA工作复现，并在目标任务上适配查看效果，需要研究人员投入数周的时间进行环境的搭建，代码仓库的理解，以及运行调试，是典型的脏活累活。因此，我们希望通过实现experiment agent，来将复现与适配的工作自动化，在已有任务的Benchmark上自动实验，选出效果最佳的工作。

Benchmark（选用MLRC-bench，7 + 3）

本次实验选择 **MLRC-Bench（Machine Learning Research Competition Benchmark）** 来考察实验型智能体（Experiment Agent, EA）在**科研导向任务**上的综合能力。该基准源自近年计算机领域的前沿的研究竞赛任务，涵盖LLM后门恢复、天气预测、商品推荐系统等多种研究方向。其设计目标是在可控环境下重现真实科研过程，通过客观的性能指标、资源约束及标准化评测脚本，系统衡量智能体在方法探索与实验执行两方面的能力。MLRC-Bench 因而为评估语言模型驱动智能体是否具备真正的“研究型实验能力”提供了更具挑战性与科学性的基准框架。

评估指标

Relative **I**mprovement **t**o **H**uman**:**

agent实现方案相对于baseline的提升与人类提升的比值，用于衡量agent与人类最高水平的差距。

$$RITH = \frac{s_{agent} - s_{baseline}}{s_{top_human} - s_{baseline}} \times 100\%$$

相关工作

Agent的本质是"LLM+循环+工具调用"。在工具调用层面，诸如 MLAB 和 Data Interpreter 等开创性系统在线性工作流中使用 LLM，通过工具自主完成数据驱动任务。OpenHands 提升了工具使用的稳定性并提供了丰富的工具集。MLZero 为智能体配备预定义的 ML 库，以进行算法选择与适配。然而，这些基于链的方式沿单一线性路径迭代，缺乏多样化的解法探索，且容易导致错误累积。

在循环层面，为实现更广泛的解空间探索，研究者将树搜索算法与智能体结合。AIDE 使用简单规则进行节点扩展或改进。SELA 在粗粒度的 ML 阶段（例如预处理、特征工程）上组织搜索，依赖仅在

模型训练后才能获得的稀疏任务级性能反馈。R&D-Agent 会并行探索由智能体生成的想法，但受制于硬编码的算法可用性，限制了其产生高质量解法的能力。ML-Master 是当前在 MLE-Bench 上表现最好的方法，采用带有强化推理组件的 Monte Carlo Tree Search (MCTS)。

在完成研究任务的范式上，有"平地起高楼"与"站在巨人的肩膀上"两种。现有工作大多让LLM从零开始写一个代码文件，即平地起高楼。虽然能正常运行，但代码简单，与人类最高水平差距较大。站在巨人的肩膀上，即利用现有的任务相关的论文或代码仓库。MLZero预先为每类任务配置了若干ML仓库，RepoMaster利用github中搜索到的与目标最相关的代码仓库去完成任务。但对于复杂的代码仓库，agent往往理解困难，无法很好的利用，甚至会随意修改，化简原代码仓库。

Level 0 基础Agent

关注点：观察学习OpenHands的“LLM + 循环 + 工具调用”的**基本骨架设计**，即如何将 LLM 与工具调用、执行环境结合为一个可运行的系统。以 OpenHands 为代表的工作，通过模块化架构让 LLM 能够在安全沙箱中执行指令、调用代码环境、进行文件操作或访问外部接口。结合代码与输出日志，观测事件循环是如何在“思考-调用-观察-反思”进行。通过阅读这一层次的论文，可以理解一个 Agent 从输入到输出的核心路径、工具调用的控制逻辑，以及系统在执行安全与错误恢复方面的设计，是理解更复杂智能体机制的起点。

Openhands.pdf

Level 1 Agent+搜索

关注点：这一层级的核心在于探索机制，即如何通过引入搜索过程提升智能体在任务上的表现。单一线性的调用流程往往缺乏探索能力，而结合搜索策略的智能体能够在解空间中生成、评估并选择更优方案。与仅依赖单次推理的 Agent 不同，这类方法通过不断扩展与修正搜索节点，实现了更具鲁棒性与创造性的解法生成。阅读此类论文有助于理解 Agent 如何通过**“推理—探索—反馈”的循环**来突破单步生成的局限。

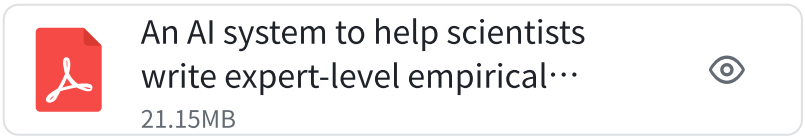
ML-Master在传统 Agent 框架上引入了带有强化推理模块的蒙特卡洛树搜索（MCTS），让智能体能够在多个候选路径间进行动态探索与回溯，从而在复杂任务上获得更高的成功率。

ML-Master.pdf
5.67MB

R&D-Agent 将系统拆分为 Researcher / Developer 双智能体：前者根据性能反馈持续提出改进思路，后者依据错误日志迭代实现，并支持多条并行探索迹线之间的信息共享与最终“融合”（Multi-Trace Merge），从而把各自的长处合成更强的解法。

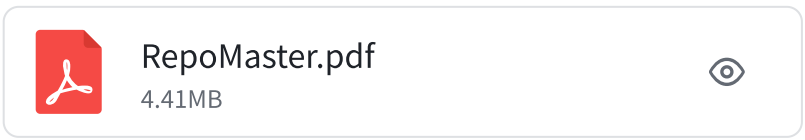
R&D-Agent Automating Data-Driven AI Solution Building···
1.55MB

Google借助“代码级变异 + Tree Search + 外部知识注入”深化探索的深度与新颖性。用 LLM 重写代码生成候选解，再用树搜索在“改写—执行—评分”的空间里系统探索，并且能将外部论文/教材/搜索结果中提炼的研究想法注入代码实现。



Level 2 代码仓库级别复现适配Agent

关注点：重点关注 Agent 如何在复杂的代码生态中完成任务，即让智能体不再从零生成，而是学会理解与复用大型代码仓库。以 RepoMaster 为代表的工作展示了这一方向的典型思路：智能体首先通过仓库解析与依赖分析，识别出与任务最相关的函数与模块，再在此基础上进行适配与增量修改，从而实现“站在巨人肩膀上”的智能复现。相比从头编写代码，这种方式更贴近真实研究与工程场景，但也对智能体的上下文理解、代码结构分析与改写能力提出了更高要求。重点去理解 **Agent 在面对真实代码库时的挑战与策略。**



挑战：不忠实的适配

Level1 通过搜索的方法去迭代优化，但输出的方案仍大多平庸，与人类的SOTA工作无法相比。而 **Level 2**虽然可以利用现有SOTA代码仓库完成任务，可是过程中Agent 常常面临理解复杂工程结构与调试执行错误的双重挑战。当遇到依赖缺失、接口不匹配或执行异常等问题时，Agent 往往倾向于以“化简”的方式绕过困难，例如删除关键模块、简化算法模型或重新实现部分功能，从而牺牲了原始方案的完整性与性能。这种行为体现了智能体在面对复杂系统时的“不忠实适配”问题，即未能忠实地继承和利用已有代码资源或者论文的思想，而是以近似的低复杂度实现代替真实复现。

1. 能否分析触发不忠实行为的因素以及背后的原因。
2. 针对这一问题，是否可以检测Agent何时发生不忠实行为，并进行干预与缓解。

通用codingAgent代表：openhands项目解构

前置知识学习

学习资料：

- openAI prompt实战
  - https://platform.openai.com/docs/guides/prompt-engineering

- 提示工程中系统消息和用户消息之间的区别
  - https://medium.com/@dan_43009/the-difference-between-system-messages-and-user-messages-in-prompt-engineering-04eaca38d06e
- 一些好的system prompt参考
  - https://github.com/x1xhlol/system-prompts-and-models-of-ai-tools.git
- Agent 的概念、原理与构建模式 —— 从零打造一个简化版的 Claude Code
  - https://github.com/microsoft/ai-agents-for-beginners
  - https://www.youtube.com/watch?v=GE0pFiFJTKo
- Agent中的ReAct设计模式
  - https://arxiv.org/abs/2210.03629

其他相关资料内容

- Chain-of-Thought（CoT） Prompting：https://arxiv.org/abs/2201.11903
- Language Models are Few-Shot Learners：https://arxiv.org/abs/2005.14165
- CodeAct：ReAct设计模式的一种实现：https://arxiv.org/abs/2402.01030
- 如何构建一个ReAct Agents：https://medium.com/google-cloud/building-react-agents-from-scratch-a-hands-on-guide-using-gemini-ffe4621d90ae
- Agnet中上下文管理（Context Engineering）是什么
  - https://www.youtube.com/watch?v=25DEMZ7wsSM
  - https://github.com/davidkimai/Context-Engineering?tab=readme-ov-file
- Tool Use-MCP和Function Calling关系与区别
  - https://www.youtube.com/watch?v=BT5tPe9dcpU

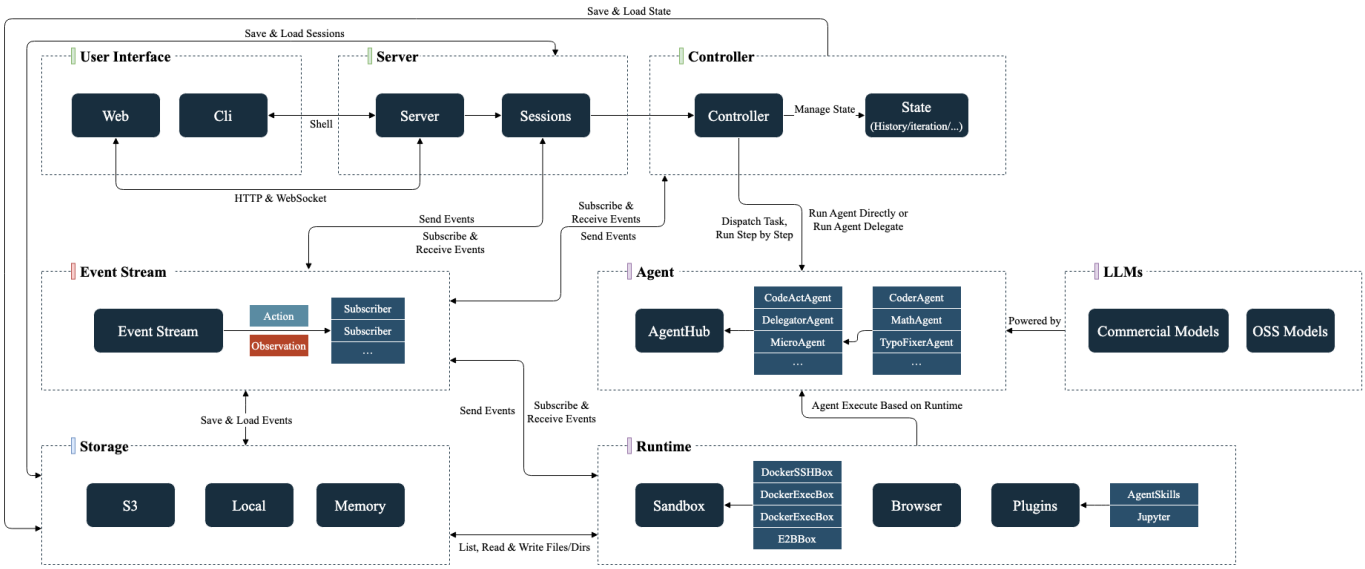
Openhands介绍

整体架构

项目地址：https://github.com/All-Hands-AI/OpenHands

详细架构：https://docs.all-hands.dev/modules/usage/architecture/backend

1 System Architecture Overview (Jul 4, 2024)



主要模块说明：

Server：用于与客户端链接做session和业务管理

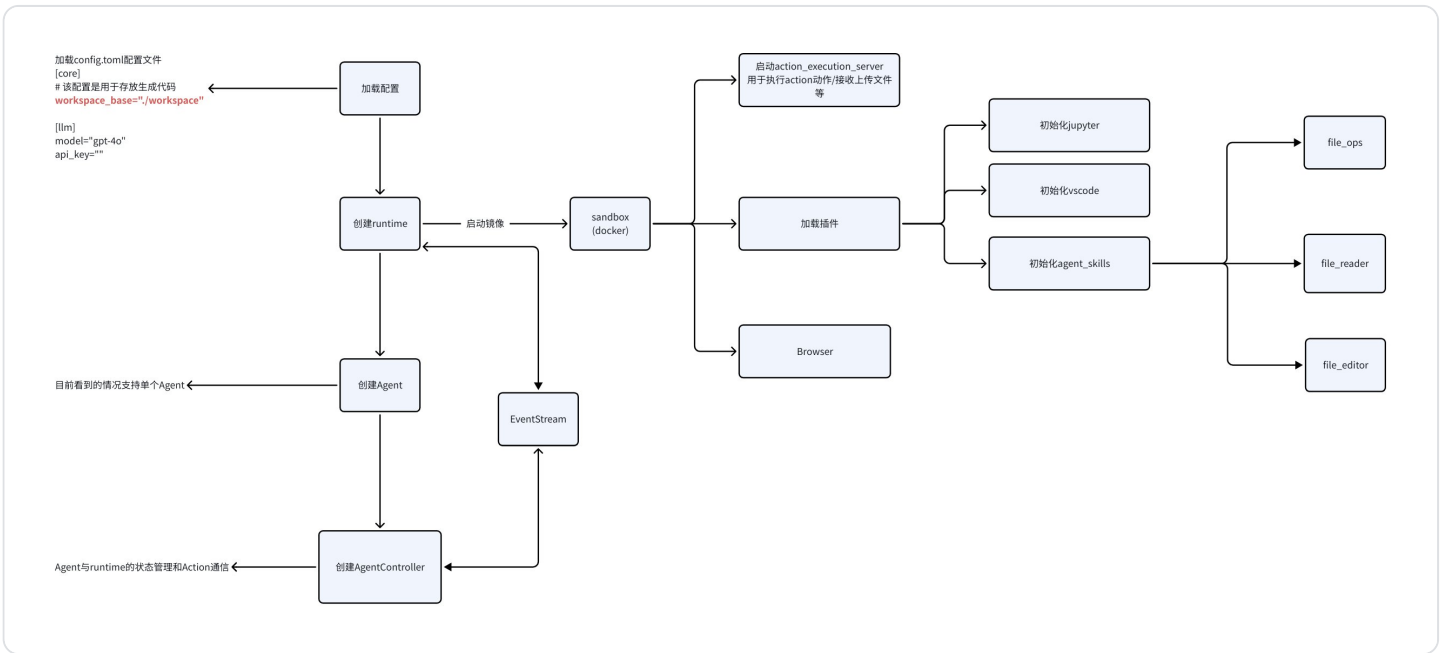
Controller：管理Agent和Runtime相关的状态，与大模型交互，上下文管理，为Agent派发Action

AgentHub：用户管理各个agent

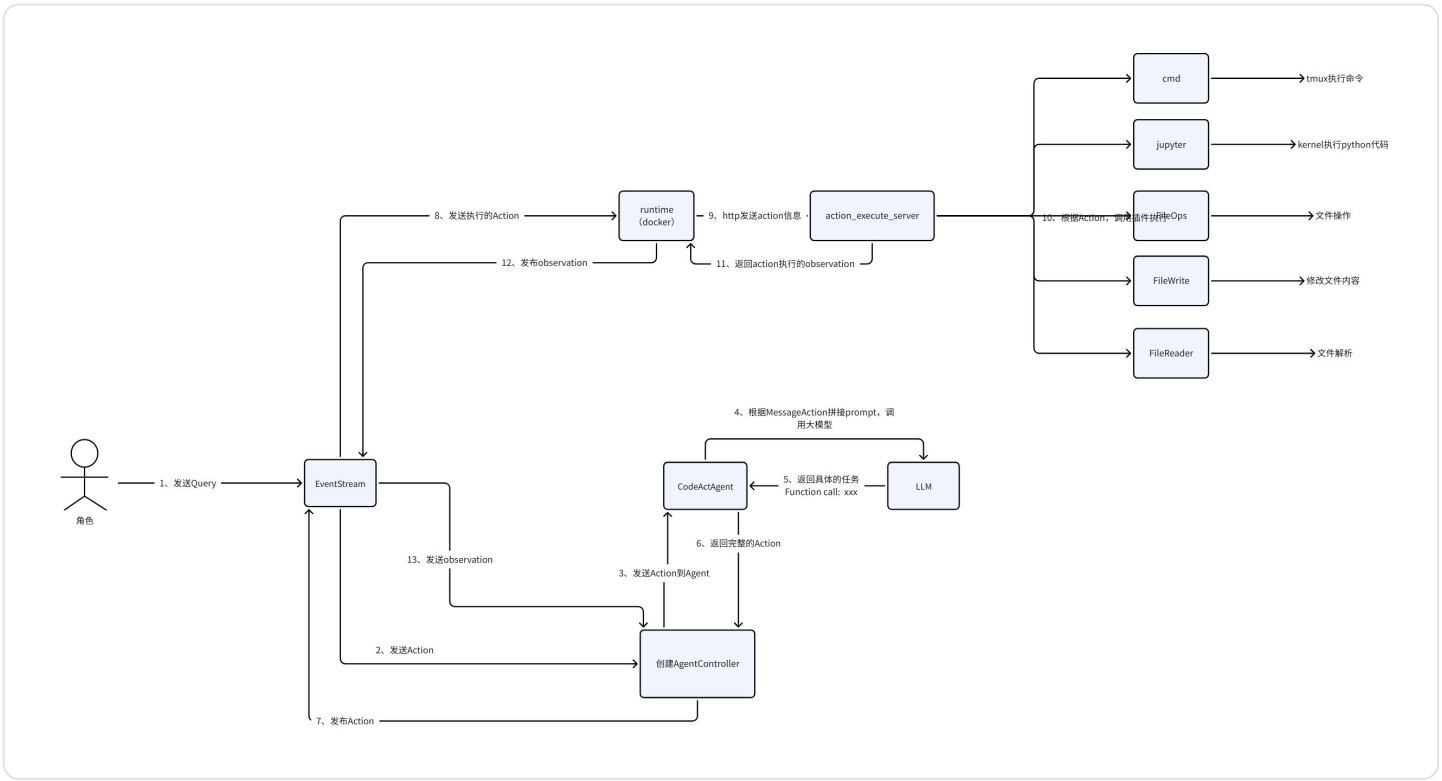
Runtime：提供和管理Action的执行环境

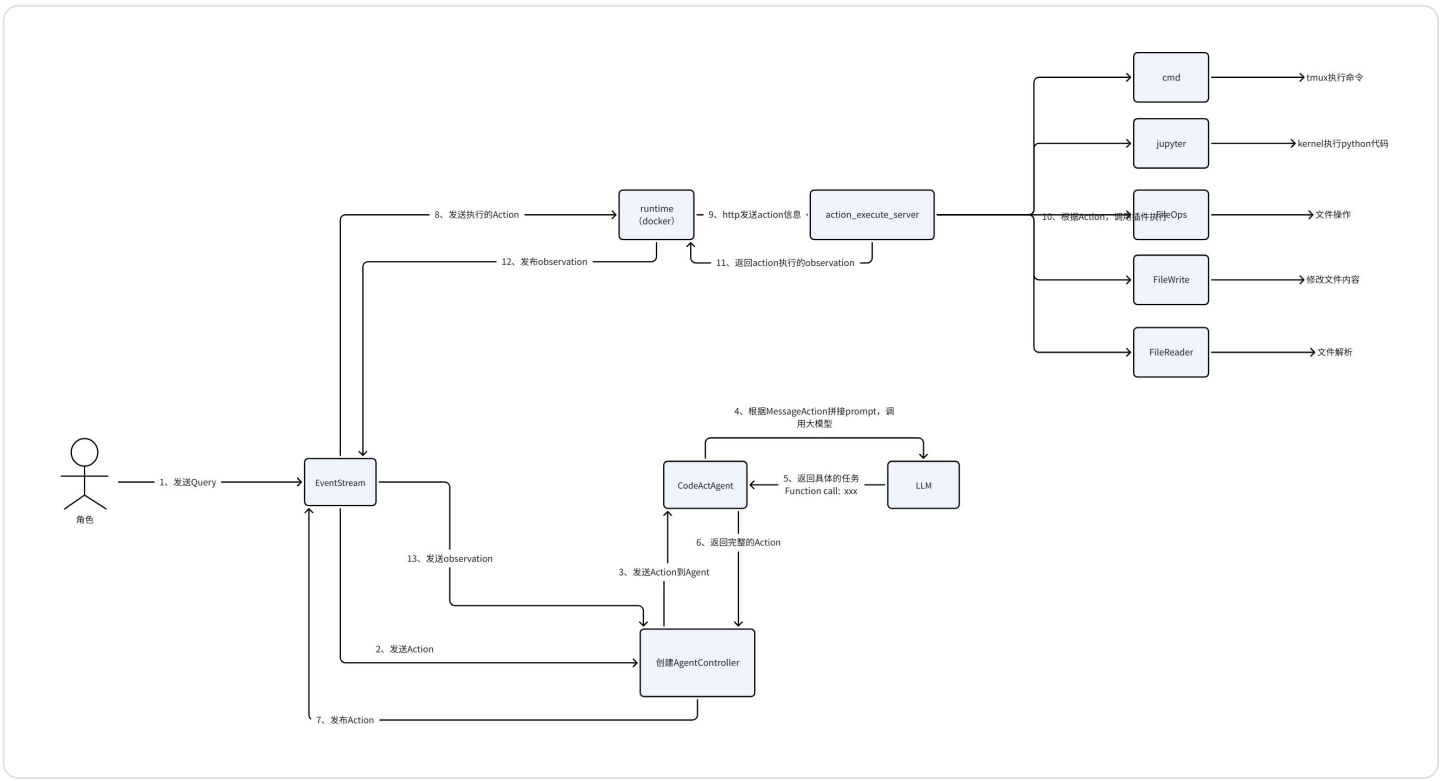
EventStream：事件中心，用于系统中事件的发布订阅用于Controller和Runtime的状态、执行结果通知

启动流程：



执行流程





Openhands中CodeActAgent的Tools：

- **Bash**： 在持久shell会话的终端中执行bash命令。
- **Ipython**：在ipython环境中运行Python代码单元(cell)。
- **llm_based_edit**：基于LLM编辑文件

- **str_replace_edit:** 文件相关操作（创建、编辑、查看），提供了字符串替换修改、查看文件行对应内容
- **Think:** 使用这个工具来思考一些事情。它不会获取新信息或对存储库进行任何更改，而只是记录这个想法在对话历史中。当需要复杂的推理或头脑风暴时使用它。
- **WebRead:** 从网页上读取（转换为MarkDown标记）内容。
- **Browser：** 使用Python代码与浏览器交互。
- **Finish:** 表示当前任务或会话完成的信号。