9.22 论文分享

《AgentOrchestra: A Hierarchical Multi-Agent Framework for General-Purpose Task Solving》

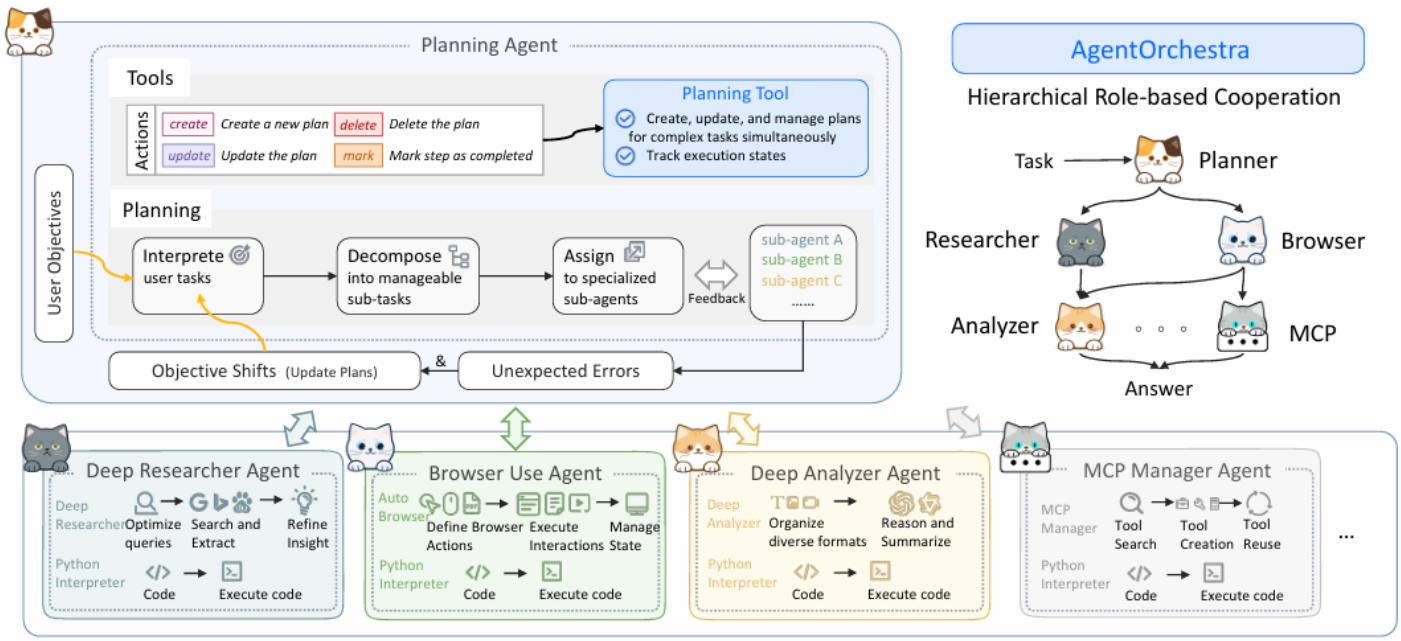


Figure 1: Architecture of AgentOrchestra.

论文核心思想

该论文介绍了一个名为**AgentOrchestra** 的分层多智能体框架，旨在解决现有基于大型语言模型（LLM）的智能体系统在通用性、多模态推理、可扩展性和协作方面面临的挑战。

该框架采用了两层架构：

- **规划智能体 (Planning Agent)**：位于顶层，负责高层次的推理、任务分解和自适应规划。它将复杂任务分解为可管理的子任务，并根据专业性将这些子任务分配给专门的子智能体。
- **专业子智能体 (Specialized Sub-Agents)**：配备了领域特定工具，负责执行具体的任务。

Map reduce 简单通用

框架的关键组成部分

该框架包含四个主要的专业子智能体，每个都专注于不同的任务类型。它们除了各自的专用工具外，都配备了 Python 解释器工具，以进行数据分析、计算验证和分析支持。

- **深度研究智能体 (Deep Researcher Agent)**：专注于全面的信息检索，通过专用的“深度研究工具”和 Python 解释器工具，进行目标性网络搜索、提取相关见解并生成内容摘要。

- **浏览器使用智能体 (Browser Use Agent)**：专注于精确和有针对性的网络信息获取，能够执行诸如网络搜索、导航、内容提取和文档操作等多种任务。
- **深度分析智能体 (Deep Analyzer Agent)**：用于高级数据分析和解释任务，能够处理文本、图像、音频和视频等多种数据格式。
- **MCP 管理智能体 (MCP Manager Agent)**：通过自动创建、动态检索和系统性重用 MCP（Model-Context Protocol）工具，实现智能体的自我进化能力。

样例

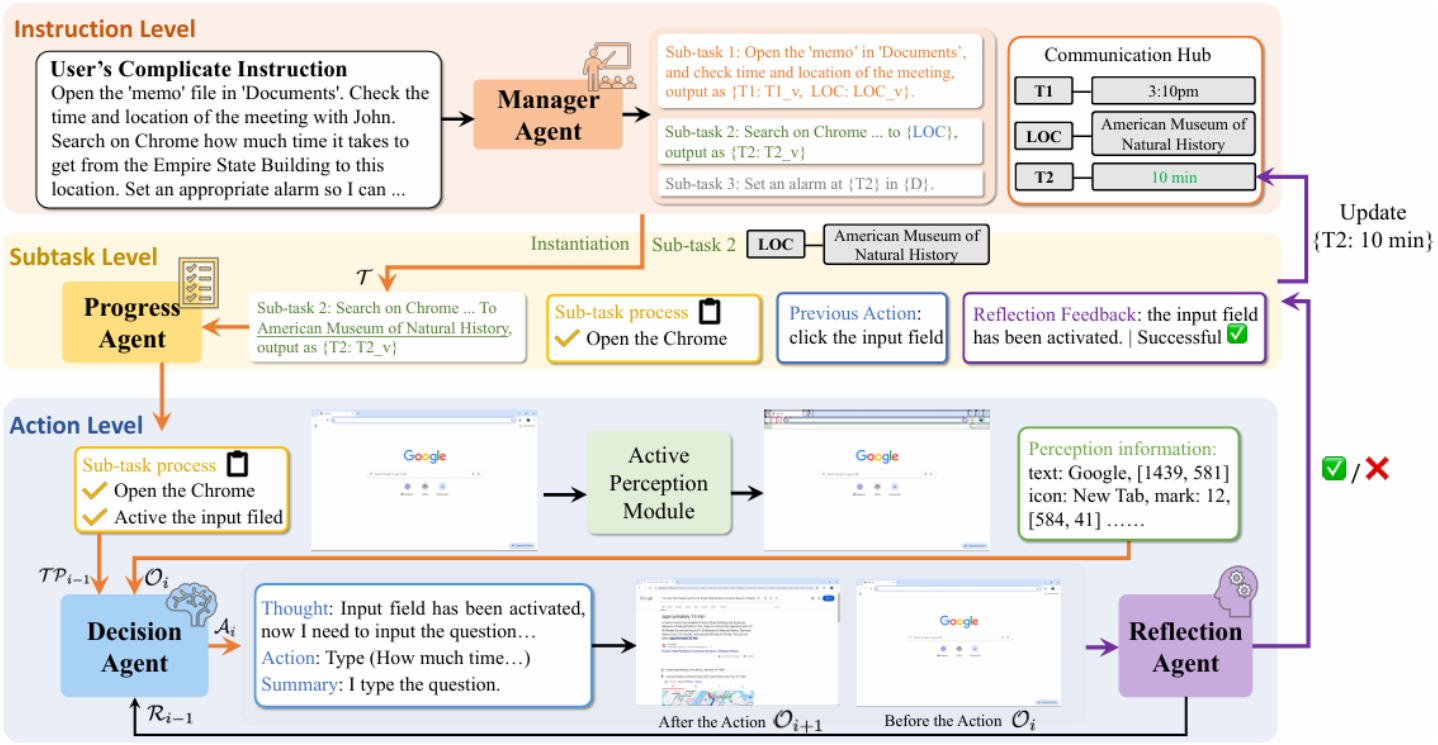
计算 Eliud Kipchoge 以其创纪录的马拉松配速穿越地球与月球之间最短距离所需的千小时间隔数。该任务的难度等级为1，不需要额外文件，并且依赖于代理的互联网信息检索、浏览器导航和计算分析能力。

AgentOrchestra 首先生成一个计划，然后通过调用子 agents 来顺序执行该计划。browser_use_agent 随后获取关键信息，包括 Eliud Kipchoge 的马拉松世界纪录（2小时01分09秒，柏林马拉松，2022年9月25日）以及月球最近点的最低距离（356,400公里）。在收集这些信息后，deep_analyzer_agent 进行必要的推理和计算，以得出答案，即17（四舍五入到最近的千小时）。值得注意的是，AgentOrchestra在获得结果后也会进行必要的验证步骤，如计算检查和基于互联网的验证。

不足

- **子智能体间通信未明确：** 论文主要描述了规划智能体与子智能体之间的自上而下的通信模式，但并未详细说明不同专业子智能体之间是如何直接协作和交流的。
- **中心化依赖：** 整个框架高度依赖于顶层的**规划智能体**。如果规划智能体在任务分解或决策上出现偏差，可能会影响整个系统的效率和准确性。（是否需要中心化planning？怎么做planning？怎么评估？）
- **潜在的复杂性：** 尽管分层架构带来了优势，但一个包含多个专门智能体的系统在设计、实现和维护上可能会比单一的智能体系统更加复杂。

《PC-Agent: A Hierarchical Multi-Agent Collaboration Framework for Complex Task Automation on PC》



论文核心思想

本文旨在解决多模态大语言模型（MLLM）在PC端自动化复杂任务时所面临的挑战。由于PC环境互动复杂、应用内及跨应用工作流繁琐，现有的单一智能体方法难以有效应对。因此，论文提出了一个名为 **PC-Agent** 的分层智能体框架，通过**分层多智能体协作**、**主动感知**以及**基于反思的动态决策机制**，显著提升了在复杂任务上的成功率。

框架的关键组成部分

主动感知模块 (APM)：

- 用于增强对屏幕上交互元素和文本的感知和操作能力。
- 通过可访问性树（accessibility tree）获取交互元素的位置和描述。
- 通过意图理解智能体（intention understanding agent）和光学字符识别（OCR）工具，精确提取和定位文本信息。

分层多智能体协作：

将决策过程分解为指令（Instruction）、子任务（Subtask）和动作（Action）三个层次。

- **管理者智能体 (Manager Agent, MA)**：位于指令层，负责将复杂指令分解为参数化的子任务，并管理子任务之间的依赖和通信。
- **进度智能体 (Progress Agent, PA)**：位于子任务层，负责跟踪并总结每个子任务的执行进度，避免冗长的历史信息干扰决策。
- **决策智能体 (Decision Agent, DA)**：位于动作层，根据APM的感知信息和PA的进度信息，生成每一步的具体动作决策

基于反思的动态决策机制：

- 引入一个 **反思智能体 (Reflection Agent, RA)**，与决策智能体并行工作。
- RA在动作执行前后观察系统状态，判断动作是否成功，并提供及时反馈。
- 这种机制使得智能体能够及时发现并纠正感知和决策中的错误，避免陷入无意义的重复或错误步骤。

样例

论文以一个包含跨应用工作流的复杂指令为例进行了说明 ()。

用户指令： “打开‘文档’中的‘备忘录’文件。查看与约翰开会的时间和地点。在Chrome上搜索从帝国大厦到此地需要多长时间。设置一个适当的闹钟，以便我...” 。

PC-Agent的执行过程：

1. **管理者智能体**将此复杂指令分解为多个子任务，如“打开备忘录文件”、“在Chrome上搜索”等。
2. 在执行过程中，**管理者智能体**维护一个通信中心，将已完成子任务（如会议地点）的输出更新到中心，并用于实例化后续子任务，例如搜索子任务。
3. **决策智能体**在子任务层面执行具体的点击、输入等操作，例如“点击输入框” 。
4. **反思智能体**检查操作结果，例如反馈“输入框已被激活。成功” 。

启发

- 在 **Controller Agent** 与 **Log/Trace/Metric Agent** 之间引入类似 **Progress Agent**，负责跟踪并总结 **Log/Trace/Metric Agent** 的执行进度，避免冗长的历史信息干扰决策。
- 增加**反馈机制**，如 **Metric Agent** 分析 Metric 数据后能否定位到故障组件，若能，则终止后续操作；若不能，则启用其他如 **Log Agent** 分析 Log 数据定位故障组件，减少冗余操作。

设计整体结构，retrieval agent，broser agent