MAESTRO：用于测试、可靠性和可观测性的多智能体评估套件

Tie Ma*† 北京航空航天大学
Yixi Chen* 阿卜杜拉国王科技大学
Vaastav Anand MPI-SWS
Alessandro Cornacchia 阿卜杜拉国王科技大学
Amândio R. Faustino 阿卜杜拉国王科技大学
Guanheng Liu 北京航空航天大学
Shan Zhang 北京航空航天大学
Hongbin Luo 北京航空航天大学
Suhaib A. Fahmy 阿卜杜拉国王科技大学
Zafar A. Qazi 拉合尔管理科学大学和阿卜杜拉国王科技大学
Marco Canini 阿卜杜拉国王科技大学

arXiv:2601.00481v1 [cs.NI] 1 Jan 2026

摘要

基于大语言模型（LLM）的多智能体系统（MAS）正迅速从演示阶段迈向生产环境，但其动态执行过程具有随机性，容易发生故障，也难以复现或调试。现有基准测试大多侧重于应用层结果（例如任务成功率），对执行行为的观察能力有限且缺乏标准化，因此难以对异构多智能体系统架构进行受控、公平的比较。

我们提出 MAESTRO，这是一个用于测试、可靠性和可观测性的基于大语言模型的多智能体系统评估套件。MAESTRO 通过统一接口标准化多智能体系统配置与执行，并借助示例库和轻量级适配器支持集成原生及第三方多智能体系统，同时导出与框架无关的执行轨迹以及系统级信号（例如延迟、成本和故障）。我们以 12 种具有代表性的多智能体系统实例化 MAESTRO，涵盖流行的智能体框架和交互模式，并针对重复运行、后端模型和工具配置开展受控实验。案例研究表明，多智能体系统执行在结构上可能保持稳定，但在时间上存在差异，导致性能和可靠性在不同运行间出现显著方差。我们还发现，多智能体系统架构是影响资源配置、可复现性以及成本–延迟–准确率权衡的主要因素，其影响往往超过后端模型或工具设置的变化。总体而言，MAESTRO 支持系统化评估，并为智能体系统的设计与优化提供实证指导。

1 引言

基于大语言模型的多智能体系统（MAS）能够灵活地解决任务，以极少修改底层系统架构的方式处理多样化和多模态工作负载 [20] 。然而，这种灵活性也给系统负载和执行行为带来了很大的不确定性。不同于传统的确定性工作流，基于大语言模型的多智能体系统采用动态执行模型，在该模型中，决策由大语言模型的输出驱动，并在运行时即时做出，而不是遵循静态定义的控制流。

重要的是，不应将多智能体系统仅仅视为一组轻量级客户端框架。相反，它们是复杂系统，其特点是动态交互 [53]、涌现行为 [20], 以及广泛的故障模式 [5]。这些特征对可预测性、可观测性和性能隔离等传统假设提出了挑战，使得传统的系统优化技术在这种情况下效果不佳。因此，一套能够系统地刻画多智能体系统执行行为的基准测试套件，对于寻求性能优化的系统运维人员以及旨在发现开放性挑战和创新机遇的研究人员而言，都至关重要。

遗憾的是，现有的基于大语言模型的多智能体系统标准化基准测试仍然有限，并且通常缺乏对多智能体系统执行行为的广泛覆盖。以往的工作主要聚焦于大语言模型服务和推理效率 [3, 7, 23, 31, 32, 50], ，评估服务器端模型性能，而不是智能体系统的执行行为。随着基于大语言模型的多智能体系统的出现，近期的基准测试 [2, 4, 15, 22, 26, 35, 47, 49, 53, 56] 开始评估单个智能体的能力（例如工具使用和通信策略）；然而，它们大多仍集中于应用层性能（例如任务成功率和响应质量），未能从标准化、全面的可观测性视角，提供对多智能体系统执行的系统级影响及相应工作负载管理挑战的评估。这种碎片化使得人们难以分析复杂的运行时行为，也难以在不同设置下对系统进行一致比较。与这一不足相一致，近期一项调查 [41] 报告称，近 75% 的运营生产环境多智能体系统的团队在没有基准测试的情况下评估其系统，而 25% 的团队构建自定义基准测试，从而限制了跨场景的可移植性和复用。

基于这些观察，我们定义了评估基于大语言模型的多智能体系统的优质基准测试所需的以下核心目标：

O1：架构异构性。基于大语言模型的多智能体系统的执行栈具有很强的可塑性。同一目标可以通过多种配置实现，包括智能体数量、角色分配、交互拓扑（例如集中式、层级式或点对点）以及通信协议。此外，设计空间还涵盖编排框架、后端大语言模型、预算约束、工具可用性和记忆机制等方面的选择，以及反思和终止策略。

O2：功能代表性。智能体工作流和现实世界部署的迅速激增，促使多智能体系统架构日益多样化，其中许多架构针对

*贡献相同。
†Tie Ma 在阿卜杜拉国王科技大学实习期间完成的工作。

表1： **案例研究中的系统性发现总结（§4）**

主题	Ref.	发现
资源	§4.2	执行仅需极少资源：不到 1 GB 的内存、 <20% 个 CPU核心，以及 MB 级流量。
通用	§4.3	交互结构保持稳定，而调用序列则表现出时间不稳定性。
	§4.7	工具集成可缓解推测性生成，从而降低延迟和成本。
后端	§4.5	模型扩展带来的收益并不一致；执行动态对性能起主导作用。
	§4.6	模型特定故障会因执行动态而显著加剧。
架构	§4.2	多智能体系统架构显著影响资源消耗特征。
	§4.3	架构决定调用图相似度，并影响系统可复现性。
	§4.4	广义架构会带来更高的资源开销，却无法提升准确率。
	§4.7	准确率的提升取决于架构，并以较低的执行开销为前提。

特定任务模式或应用领域进行了优化。近期设计探索了日益复杂的协调与推理策略 [45, 52, 57, 58]。因此，没有任何单一架构能够代表更广泛的多智能体系统设计空间。

O3：执行可追溯性。 当前商业智能体系统通常会公开高层次的推理轨迹，但对执行级细节和系统内部状态的呈现有限，且缺乏统一标准。此外，现有多智能体系统模块缺少统一的遥测标准，往往导致信息被“静默”消耗：不同的大语言模型提供商和框架未能向用户公开关键运行数据 [44]。

为弥补这一不足，我们推出 MAESTRO，这是一个面向基于大语言模型的多智能体系统的综合性开源评估套件。
MAESTRO 旨在对不同智能体架构、交互模式和运行时条件下的执行行为进行系统性表征，从而为有据可依的系统优化提供参考。

我们的贡献主要有三方面：
丰富且可扩展的基准测试。 MAESTRO 纳入了 12 个具有代表性的多智能体系统示例，每个示例都具有鲜明的架构差异，可作为系统性提炼洞见的基础，如表1所示。此外，
MAESTRO 具备可扩展性，社区只需付出极少努力，即可在我们的评估框架中集成并复用现有的多智能体系统实现。

与框架无关的系统集成。MAESTRO 基于一系列广泛使用的开源智能体框架及示例构建 [10, 17, 25], , 旨在捕捉实践中常见的架构模式，而非偏向某一种工作流设计。

统一的执行级遥测标准。MAESTRO 定义并实现了统一的遥测接口，旨在全面采集不同模块的执行数据。该架构建立了多智能体系统各组件均可遵循的通用协议，从而确保在整个系统生命周期中进行一致且透明的监控。

MAESTRO 可通过 https://github.com/sands-lab/maestro 获取。

2 背景

2.1 基于大语言模型的多智能体系统剖析

基于大语言模型的多智能体系统（多智能体系统）由多个协同工作的 大语言模型智能体组成，用以完成超出单个智能体能力范围的大型任务 [21]。在典型的多智能体系统中，多个专用智能体协同规划、协调并执行大型任务，每个单个智能体则专注于特定的子任务。

构成模块。大语言模型智能体是一种能够自主执行多步骤任务的实体，它结合生成式基础模型、外部工具、内存以及推理和规划能力 [33]。智能体旨在高度动态的环境中自主运行，在这些环境中，适应能力和战略决策至关重要。每个智能体由四个关键部分组成：（i）输入，可能包括用户指令、开发者指定的约束、多模态观测、检索到的知识和内部状态；（ii）生成式大语言模型（大语言模型），将当前状态映射为决策；（iii）动作接口，可支持数据检索、API 调用和代码执行等工具交互；（iv）输出，包括面向用户的响应、结构化动作和工件，以及更新后的状态。

编排与部署。从业者通过使用智能体编程框架编写的工作流来编排多智能体系统，例如 LangGraph [25], CrewAI [24], Autogen [10], LlamaIn- dex [36], 和 Agno [37]。尽管根据调查 [39, 41], , 这些第三方框架广受欢迎，但对从业者的详细访谈表明，在 85% 的情况下，从业者更倾向于从头构建智能体应用 [41]。根据开发者赋予这些系统的自主性程度，工作流可以是静态的，也可以是动态的。目前，多智能体系统以单体应用的形式部署；不过，它们正越来越多地以分布式应用的形式开发和部署 [9, 46]。

工作流结构。 多智能体系统工作流通常遵循层次结构，其任务结构呈现为由子任务组成的树。单个子任务遵循顺序流与并行流相结合的模式。工作流还可能包含针对单个智能体的递归调用 [39]。

故障类型。多智能体系统应用展现出三种主要故障类型——系统设计问题、智能体间失调、任务验证 [5]。系统设计问题包括配置问题、API 和系统问题，以及资源管理不善。智能体间失调问题源于执行期间智能体间交互与协调导致关键信息流中断。这包括规划与协调错误、错误的输出生成、单个大语言模型幻觉，以及错误的信息处理。当验证策略不足以识别问题时，就会出现任务验证故障。

非确定性的来源。 由于多智能体系统应用具有动态且异构的特性，因此会因诸多原因表现出非确定性。首先，大语言模型本质上具有随机性，往往会对相同输入生成不同输出。其次，外部工具执行并非预先规划或编程确定的。此外，工具可能产生非确定性结果。第三，工作流是动态的，并会在运行时发生变化 [53]。动态工作流中的非确定性还可能因智能体的可用性而进一步加剧。第四，内置的可靠性机制会影响多智能体系统执行的性能和结构。例如，质量驱动的重试会改变执行图。

可靠性是首要考量。 典型的多智能体系统应用在设计和实现过程中，将可靠性视为首要考量。它们通过多种方式实现这一点。首先，大多数多智能体系统应用都依赖人在回路评估；在寻求人在回路评估之前，近半数应用执行的步骤不到五步 [41]。此外，开发人员常通过加入大语言模型评审来自动执行质量检查。如果质量检查失败，多智能体系统应用还会自动重试以提升质量。其次，从业者优先考虑质量而非实时响应能力；在最近的一项调查中，66% 的受访者表示可以接受超过一分钟的响应时间 [41]。第三，从业者倾向于采用静态工作流而非动态工作流，以限制已部署智能体的自主性 [41]。

2.2 现有基准测试的局限性

评估和基准测试大语言模型的性能，一直是衡量大语言模型执行现实世界任务时效率与效能的重要方面 [3, 23, 31, 32]。近年来，随着大语言模型智能体和多智能体系统应用的兴起，对智能体系统性能进行基准测试也日益受到科学界的广泛关注。

智能体基准测试。 典型的智能体基准测试会评估单个大语言模型作为智能体的能力 [4, 27, 35]。此类基准测试随后也扩展到了多智能体场景。为此，研究人员开发了专门的基准测试，用于评估智能体系统的特定属性，例如工具调用 [22, 56], 任务规划 [15], 通信策略 [53], 顺序流 [2], 隐私保护 [26],和协作效能 [47, 49]。此类专门的基准测试仅关注多智能体系统应用的某一特定属性或维度，缺乏全面视角，因而无法有效了解多智能体系统应用的端到端涌现行为和性能。

定制基准测试。 由于缺乏标准化且多智能体系统设计空间多样，多智能体系统应用开发者转而选择为自己的应用创建专用的定制基准测试。例如，Autogen [51] 的作者为使用 Autogen框架开发的任务创建了一个名为 AutogenBench [1] 的定制基准测试。根据近期的一项调查，生产级多智能体系统应用的团队中，25%会为其应用构建定制基准测试，75%的团队则不使用正式基准测试来评估其智能体，而是依赖 A/B测试以及专家或用户的直接反馈 [41]。尽管这些基准测试适用于特定应用，但无法涵盖多智能体系统设计空间的多样性，也无法提供广泛情境下的见解。

可观测性工具和基准测试。 Opik [8], 、TRAIL [11], 和 TAMAS [39] 等可观测性工具与基于可观测性的基准会捕获多智能体系统执行的跨度和追踪数据，开发者可利用这些数据进一步分析追踪信息、排查问题并了解多智能体系统执行情况。除了智能体调用频率、外部API使用情况和单次调用令牌成本等标准指标外，深层应用语义遥测仍存在显著缺口。要满足这一需求，需要捕获更细粒度的重试逻辑详情（例如尝试次数、触发条件和父跨度ID）、智能体特定状态条件（例如故障分类和错误推理），以及输出质量评估。此类遥测对于提供执行级透明度至关重要，有助于诊断随机性故障并理解复杂的多智能体交互。

3 MAESTRO

我们提出 MAESTRO——用于测试、可靠性和可观测性的多智能体评估套件，作为评估基于大语言模型的多智能体系统的综合框架。在 §1 中概述目标的基础上，我们将在 §3.1 详细介绍该框架的架构与设计，并说明如何调整独立的多智能体系统实现并将其集成到套件中。为展示 MAESTRO 生成信息丰富的遥测数据的能力，我们提供了一组具有代表性的多智能体系统实例（即基准测试套件中的具体评估单元）。这些实例按照我们提出的分类体系进行归类（§3.2）；§3.3 则详细介绍所使用的具体实例，以及用于得出实验发现的评估套件的构建方式。

3.1 基准设计

*3.1.1 MAESTRO*架构。 图 1 展示了 MAESTRO 架构的概览。从概念上讲，MAESTRO 遵循线性控制流：准备多智能体系统实例，用户定义的配置指定如何实例化和执行多智能体系统，在运行时收集执行轨迹，随后通过事后处理将这些追踪数据转换为可解释指标和摘要。该工作流由五个核心组件组成：

多智能体系统实例准备。 要使用 MAESTRO，用户首先需要准备待评估的多智能体系统实例。准备流程的详细信息以及支持的集成模式见 §3.1.2。

配置。 基于已准备好的多智能体系统实例，用户指定评估设置，包括输入来源、智能体实例的数量和配置，以及是否启用外部工具

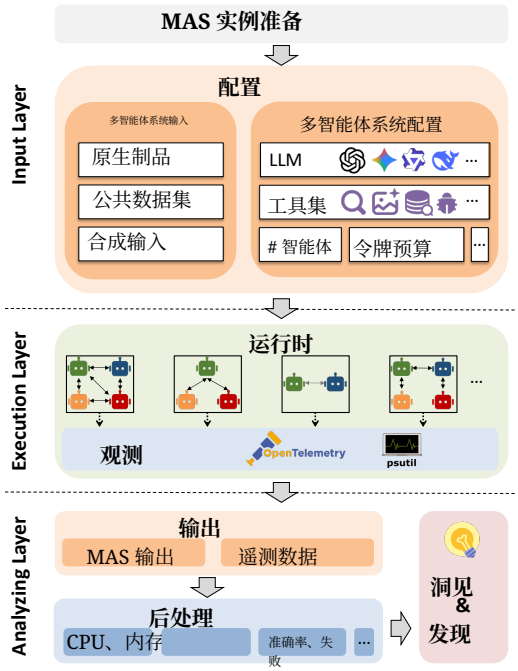


图 1：**MAESTRO架构概览。**

访问[1]。此配置将传递给运行时 组件，用于系统实例化和执行。

运行时。根据提供的配置，运行时组件负责协调多智能体系统实例的执行。任务输入会持续送入运行时，从而触发智能体交互、工具调用和控制流决策，具体方式由配置定义。

观测。执行期间，观测组件通过函数调用钩子或基于采样的监测埋点监控系统行为。该组件基于 OpenTelemetry [40] 和 Psutil [43], 构建，既记录默认执行指标（例如延迟、令牌用量；参见 §A.2.1），也记录配置中指定的其他信号。收集到的追踪数据会在线或离线转发至后处理组件。

后处理。后处理组件汇总并分析执行轨迹，使多智能体系统的行为可供检查（例如 CPU、调用图；参见 §A.1）。这些摘要使用户能够探索执行轨迹、比较不同运行中的配置，并识别性能瓶颈和不稳定性的来源。

3.1.2 多智能体系统实例准备。在执行任何评估之前，必须将多智能体系统实例集成到 MAESTRO 中。如图 2 所示，用户可以通过两种方式准备多智能体系统实例：[2]

• **MAESTRO 原生方式。**用户可以直接使用 MAESTRO 的原生规范语言和配置接口来实现多智能体系统实例。此模式利用基于编译的技术 [9]，从高层描述自动生成可执行实例

[1]目前，MAESTRO 仅支持调整少量参数，例如模型选择和工具使用。[2]目前，MAESTRO 仅支持预定义的多智能体系统实例。

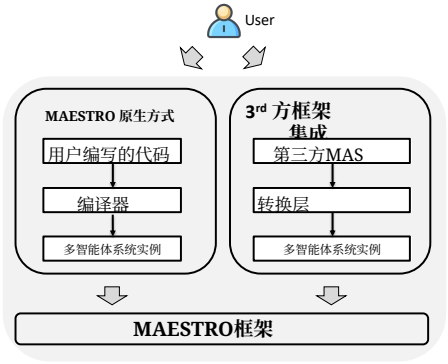


图 2：**为 MAESTRO 准备多智能体系统实例的两种方式。请注意，MAESTRO 自带一组内置的多智能体系统实例，可直接使用和比较。**

，确保最佳兼容性。此模式通过为通用组件生成可重复使用的脚手架和集成代码，减少手动编码工作。

• **第三方框架集成。**通过 MAESTRO 的转换层，用户可以在自己偏好的智能体框架（例如 ADK、LangGraph、AutoGen）中构建多智能体系统实例，或导入现有的开源实现，并将其连接到 MAESTRO 进行评估。转换层提供了一组适配器，可将特定框架的组件映射到 MAESTRO 的标准接口，从而为配置、执行和遥测数据收集提供统一的入口。

MAESTRO 的贡献者也可以使用这两种集成模式，将新的内置多智能体系统实例添加到框架中。目前，MAESTRO 内置了 12 个多智能体系统实例（详见 §3.3），用户无需额外集成工作即可直接进行评估和比较。

3.2 多智能体系统实例分类体系

设计良好的基准测试应涵盖广泛（**O1**）且具有代表性（**O2**）的系统配置和用例；否则，得出的结论可能会过度拟合多智能体系统设计空间中的狭窄片段，无法泛化。为了实现系统性覆盖和受控比较，我们采用一组定义明确的少量维度来描述每个多智能体系统实例。具体而言，我们从以下几个方面描述每个实例：应用领域、框架、交互模式和数据规范。这些维度共同涵盖了现代多智能体系统部署的主要差异来源。

应用领域。多智能体系统实例所面向的高层领域，它可能会影响任务复杂度、所需的智能体能力以及评估标准。常见领域包括问答、创意生成、金融等。

框架。 用于实现多智能体系统实例的底层多智能体框架，这可能会影响智能体编排、通信协议和工具集成。常见框架包括 AutoGen [51], ADK [17], LangGraph [25], 等。

交互模式。 多智能体系统实例中智能体的具体配置，包括智能体数量、工具数量

以及合作类型。具体而言，合作类型包括：

• 规划。有一个专门的规划智能体负责将任务分解为子任务，并分配给其他智能体。• 协调。智能体通过明确的通信来协调各自的行动。

• 辩论。智能体评估并比较候选解决方案，以达成共识。

• 纠正。智能体通过迭代反馈协作完善和改进特定解决方案。

这些交互模式可能会影响整体系统动力学和性能。

数据规范。 用于实例化多智能体系统实例的具体输入输出格式和真实标注，这些内容可能会影响任务复杂度、通信模式和评估标准。数据规范可分为输入和输出。

• 输入。 任务通常通过系统提示来实例化，系统提示用于定义任务的核心目标和约束。对于需要开放式探索的任务，配置阶段还可能将通过辅助工具（如网络搜索引擎或私有数据库）检索到的外部信息作为补充输入。我们将这些结构化输入和检索到的数据统称为工件。 • 输出。 根据输出是否具有确定的真实标注，可将输出分为开放式和封闭式。

3.3 已研究的多智能体系统示例套件

我们精心挑选了 12 个具有代表性的多智能体系统实例，作为 MAESTRO 中预先定义的评估集。这些实例旨在充分覆盖常见的多智能体系统配置和用例（**O1, O2**），并为后续研究提供基线。如表 2 所总结，所选实例依据以下标准确定：

• **框架多样性（O2、O3）**：我们纳入了使用不同热门多智能体系统框架实现的示例，例如 *MCP*-智能体、*LangGraph*、*ADK* 和 *Autogen*，以涵盖广泛的设计模式和交互范式。

• **官方来源（O2）**：我们收集了这些框架官方示例代码库或教程中提供的示例，确保其体现最佳实践和标准用法。

• **领域多样性（O2）**：我们选择涵盖多种应用领域的示例，包括问答、规划、创意写作、营销策略等，以评估多智能体系统

在不同应用场景中的性能。
• **交互多样性 (O1)**：我们优先选择能够展现智能体之间多种交互模式的示例，例如协作问题求解、辩论式讨论和基于角色的协作，以评估不同交互风格如何影响多智能体系统的行为。

作为开展有意义的分析后处理的前提，多智能体系统实例必须按照共享相关特征的连贯类别进行分组。这样的分组能够对多个配置开展比较分析，确保观察到的行为

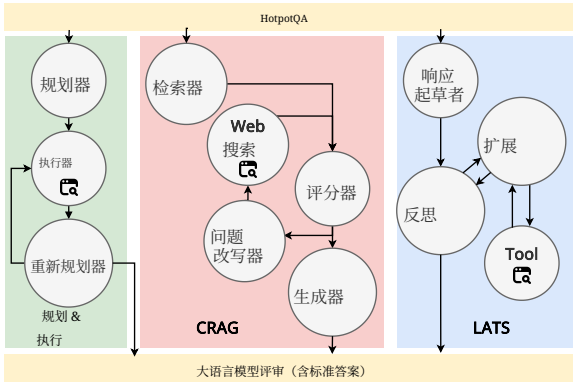


图3：**使用 3 种不同的多智能体系统架构解决相同的给定任务。**

反映系统性趋势，而非个别运行中的临时性工件。为展示 MAESTRO 的分析能力，我们构建了两个评估套件，每个套件都旨在揭示不同的系统级洞见。

• **全量套件 (F)**：该套件包含所有选定的多智能体系统示例。我们将此套件视为真实世界多智能体系统部署的代表性子集，并用它来研究基于大语言模型的多智能体系统在真实场景中的整体性能和行为。

• **架构导向套件（A）**：该套件包含三个具有代表性的多智能体系统示例，它们实现了不同的代表性多智能体架构，但解决相同的一组任务。该套件用于研究智能体架构对多智能体系统行为的影响。

在架构导向套件中，我们选取了三种具有代表性的多智能体系统架构：CRAG（纠正性检索增强生成） [54],规划与执行 [48], 以及 LATS（语言智能体树搜索） [59]。它们被设计为通用型智能体架构，能够处理各种任务，而不与特定应用紧密绑定。然而，它们的设计目标各不相同。如图3所示，CRAG 针对检索中心型工作负载进行了优化，而 LATS 和规划与执行则面向更通用的问题求解场景，分别采用基于树搜索的分治策略和贪心迭代优化策略。任务求解器架构的这种多样性，使得后续的比较研究得以开展。

在接下来的部分中，我们将围绕这两种多智能体系统示例套件，介绍使用 MAESTRO 进行的案例研究与分析。

4 案例研究

为了展示研究人员如何从 MAESTRO 中受益，我们开展了一系列案例研究，以说明其细粒度遥测能够带来哪些类型的洞见。我们将评估分为两组互为补充的案例研究：

• **通用系统级分析**。在 §4.2 和 §4.3 中，我们考察并非多智能体系统（MAS）所特有的系统级指标，而是提供一个熟悉的基准，以便像评估传统系统一样，对性能、资源消耗和可靠性进行推理。

表 2：**精选多智能体系统示例概览。“套件”列表示所属套件：F：Full Suite；A：架构套件。**

示例	应用领域	框架	交互			数据规范		套件
			Type	#Agt	工具数量	In	Out	
金融分析器 [29]	金融	MCP-Agent	正确	6	1	工件	开放式	F
图像筛选 [16]	创造力	ADK	辩论	4	2	工件	Cls-Form	F
营销 [16]	营销	ADK	协调	4	1	工件	开放式	F
品牌搜索引擎优化 [16]	营销	ADK	协调	4	10	工件	开放式	F
内容创作 [42]	创造力	ADK	规划	4	1	工件	开放式	F
Mag.-One [14]	跨领域	Autogen	Plan	4	0	工件	开放式	F
股票研究 [10]	金融	Autogen	协调	4	2	工件	开放式	F
旅行计划 [38]	旅行	Autogen	协调	4	0	工件	开放式	F
思维树 [57]	跨领域	LangGraph	辩论	3	0	工件	Cls-Form	F
CRAG [54]	跨领域	LangGraph	协调	5	2	数据集	开放式	F,A
规划与执行 [48]	跨领域	LangGraph	Plan	3	1	数据集	开放式	F,A
LATS [59]	跨领域	LangGraph	Plan	3	1	数据集	开放式	F,A

• **面向应用且具备语义感知能力的分析**。 使用评估套件 2，我们研究不同的多智能体系统（MAS）求解器架构如何影响成本、延迟和准确率（§4.4、§4.5、§4.6 和 §4.7）。此分析探讨架构选择和结构优化是否会带来一致的性能权衡，方式类似于消融研究。

4.1 方法

输入。 对于每个多智能体系统实例，我们采用以下三种方法之一生成评估输入：

• **朴素制品**。 直接复用官方示例代码库 README 文件中提供的输入提示。
• **公共数据集**。从与多智能体系统实例的任务领域相符的公开基准数据集中抽取的输入（例如，面向问答的智能体使用问答数据集 [55]）。

• **合成输入**。大语言模型生成的提示，可用于实现受控变化并提高输入多样性。

设置。 对于每个多智能体系统实例，我们至少进行 20 次独立运行，以刻画其执行行为。每次运行都包含向多智能体系统提交一个用户级任务输入（例如，为内容创作提交一条“写一篇博客文章”提示）。对于需要人在回路交互的多智能体系统实例，我们采用大语言模型作为用户方法来模拟用户回复：指定的大语言模型（在我们当前的配置中为 gemini-2.5-flash）会根据多智能体系统的输出生成回复。为避免执行不终止，每次运行的时限设为 10 分钟。对于侧重架构的套件，大语言模型回复的长度还限制在最多 8,192 个 token。关于表 2 中提到的外部工具使用情况，我们使用 Tavily 或 Google 搜索 [18] 进行网络搜索，并使用 Google imagen-3.0-generate-002 [19] 模型进行图像生成。需要评估正确性时，我们使用 gpt-4o-mini 作为大语言模型评审。为评估不同骨干模型的影响，我们在若干种配置中更换底层大语言模型：Gemini-2.0-Flash-Lite（Ge20FL）、Gemini-2.5-Flash-Lite（Ge25FL）、Gemini-2.5-Flash（Ge25F）、GPT-4o-mini（G4oM）、GPT-5-mini（G5M）和 GPT-5-nano（G5N）。

4.2 系统的使用模式及其影响是什么?

在真实系统中部署多智能体系统时，CPU、内存和网络使用量等资源消耗是需要考虑的重要因素。了解多智能体系统的资源使用模式，有助于优化其性能和可扩展性。在本小节中，我们分析多智能体系统的资源消耗，并探究影响其使用模式的因素。

在我们的设置中，每项任务的 CPU 和内存占用量都较低且有上限。图 4 展示了在我们的运行时配置下，12 个多智能体系统示例每次运行的 CPU 和内存使用情况。CPU 方面，观察到的最高利用率达到 61.9%，而大多数示例的箱线图分布都明显低于这一峰值，这表明这些工作负载通常不需要持续进行大量本地计算。内存方面，除内存占用量峰值为 1726.8MB 的内容创作示例外，各示例的平均内存使用量为 200.2MB；根据我们的测量，大多数单任务执行的内存占用都低于 1GB。内容创作示例的内存占用量较高，源于其分布式设计：每个智能体都在独立进程中运行，从而增加了驻留内存总量。我们将对分布式多智能体系统部署及其资源权衡的更广泛研究留待未来开展。我们还观察到内存使用模式因框架而异；例如，在我们的设置中，使用 *ADK* 实现的示例往往具有更高的内存占用量。

每项任务的通信量处于 MB 量级，并且因架构和模型而异。我们进一步测量了使用和不使用工具时各架构的总通信量（图5）。在所有测试配置中，观测到的通信量均处于 MB 量级（大多数情况下为几 MB），这表明每项任务的网络负载通常较小，与 CPU 和内存占用量相比

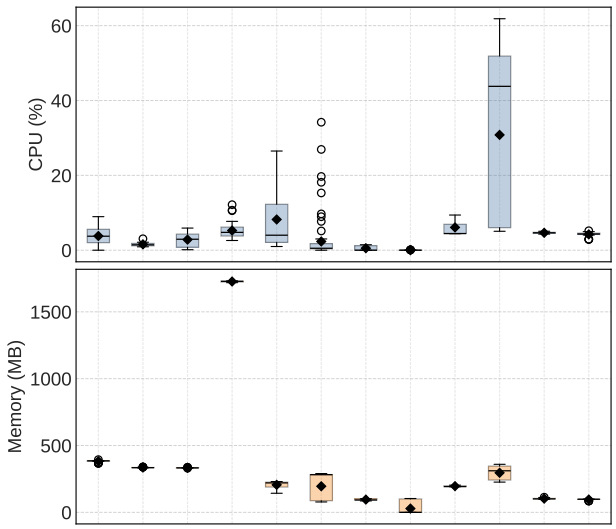


图4： **不同示例中的 CPU 和内存使用情况。**

要低。可以看出，工具的使用对通信量影响较小。模型选择也会与架构产生交互影响；例如，在我们的测量中，CRAG 中 Gemini 系列模型的通信量高于 GPT 系列模型。

发现*1*：在我们的设置中，大多数单任务执行的内存使用量保持在 1 GB 以下，CPU 使用率也处于受限范围内，而每个请求的通信量则维持在 MB 级别。

CPU 和内存使用模式取决于架构。图6展示了按架构分组的 CPU 和内存使用情况。与图5中观察到的通信模式一致，资源消耗高度依赖于架构。CRAG 的资源占用最高，平均 CPU 使用率为 9.7%，内存使用量为 405.3MB（对所有模型和工具配置取平均值）。其次是语言智能体树搜索（CPU 使用率为 1.36%）和规划与执行（CPU 使用率为 0.07%）。我们还观察到，尽管模型选择会影响 CPU 负载，但对内存的影响微乎其微。令人意外的是，启用工具后，全局平均 CPU 使用率降低了 3.1%，内存使用量减少了 4.8MB。这一现象或许可以解释为，使用工具有助于减少大语言模型调用的次数，而这些调用会大量消耗 CPU 和内存。

发现 *2*：架构主导资源使用模式，而模型选择带来的变化较小。

4.3 多智能体系统调用图有多稳定？哪些因素会影响其可变性？

与微服务等传统系统相比，基于大语言模型的多智能体系统的一个显著特征是其执行行为具有内在的随机性。在微服务架构中，调用图的变化通常被用作识别异常执行和边缘情况的指标。将这一概念扩展到多智能体系统，可以利用这种变化来抽样获得多样且具有代表性的执行轨迹。然而，此类方法的前提是进行基础稳定性分析。从可复现性的角度来看，多次运行中较高的调用图相似度意味着智能体交互在不同运行间具有更强的一致性，因此多智能体系统执行也更具可复现性。因此，量化不同运行中智能体交互的稳定性，并确定影响其变化的因素，对于设计稳健且可复现的多智能体系统至关重要。

我们使用两种指标来衡量调用图的相似度：杰卡德相似度和最长公共序列（LCS）相似度。这些指标分别捕捉调用图结构的不同方面，并为理解智能体交互提供洞见。

- **杰卡德相似度（边集重叠度）**：对于每次运行 i，我们构建一个有向调用图 G_i，并将其未加权边集记为 E_i。对于运行 i，j，我们计算

$$J(E_i, E_j) = \frac{|E_i \cap E_j|}{|E_i \cup E_j|},$$

约定 $J(E_i, E_j) = 0$ 在 $E_i \cup E_j = \emptyset$时成立。该指标用于衡量相同的交互边是否至少出现过一次，而不考虑出现频率。

- **LCS（顺序一致性）**：对于每次运行 i，我们将调用线性化为有序边序列 S_i。令 $\mathrm{LCSlen}(S_i, S_j)$ 表示 S_i 与 S_j之间最长公共子序列的长度。我们将归一化最长公共子序列相似度定义为

$$\mathrm{LCS}(S_i, S_j) = \frac{\mathrm{LCSlen}(S_i, S_j)}{\max(|S_i|, |S_j|)},$$

约定两个空序列的相似度为 1，一个空序列的相似度为 0。该指标用于衡量交互顺序的一致性。

为概括不同粒度下的相似度（例如，按示例、模型或实验条件分组，如工具开启与工具关闭），我们首先根据关注的维度将运行划分为若干组。对于包含 n 次运行的组，我们使用 $1 \le i < j \le n$ 计算所有无序运行对（i，j）的相似度。对于每一对，我们计算 $J(E_i, E_j)$ 和最长公共子序列(S_i, S_j)。我们将一组的成对平均相似度定义为该组内所有无序运行对的成对相似度均值。如果 $n < 2$，则将成对平均相似度设为 0（没有可用的成对比较）。

多智能体系统的执行展现出结构稳定性，但存在序列方差。我们首先考察同一示例多次运行时调用图的稳定性。图7展示了 Full Suite 的示例内成对平均相似度。我们发现，所有案例的 Jaccard相似度都很高（所有示例的平均值为 0.86），这表明智能体间交互的集合

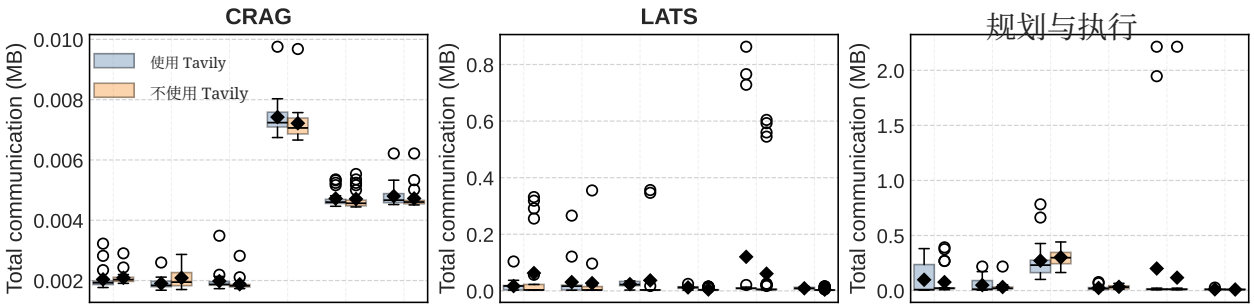


图5： **不同架构之间的通信使用情况**

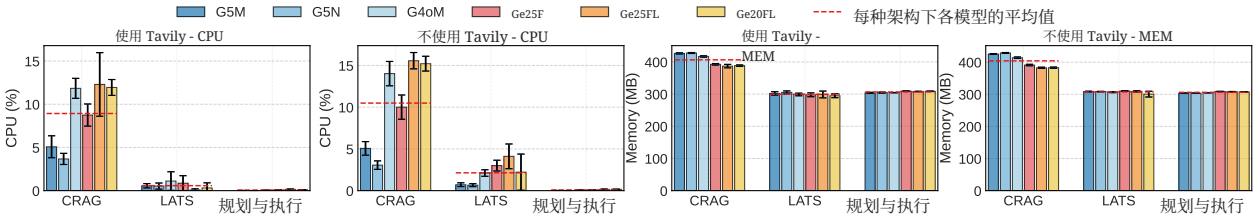
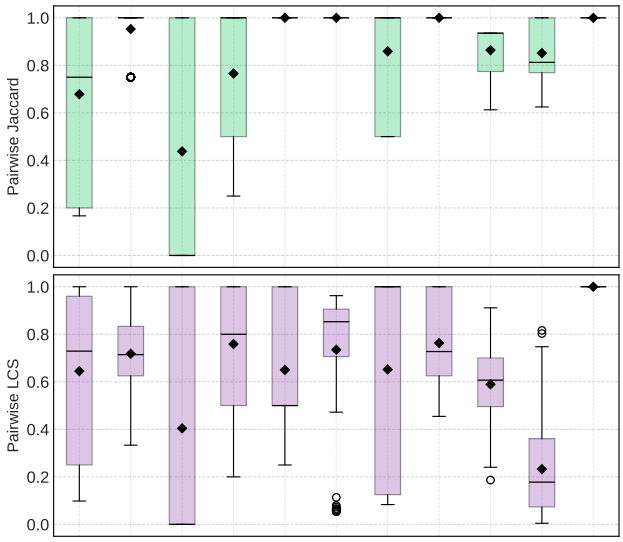


图6： **不同架构下的 CPU 和内存使用情况。**



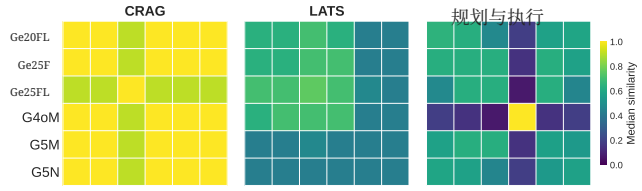


图8： **跨模型调用图最长公共子序列相似度 ty.**

图7： **跨模型调用图相似度（Jaccard相似度：不考虑顺序的智能体交互重叠度；LCS相似度：考虑顺序的执行轨迹相似度）。**

能够抵御执行方差的影响。相比之下，最长公共子序列相似度中等（平均值为 0.65），这表明序列

智能体调用 在不同运行之间波动显著。值得注意的是，CRAG 和思维树等示例的 Jaccard 分数较高而 LCS 分数较低，这表明尽管参与的智能体及其连接保持一致，但其交互的时间顺序却高度动态。一个明显的例外是旅行规划示例，它采用 *RoundRobinGroupChat* 机制（来自 *Autogen*）；该机制强制执行确定性的执行顺序，因此两项指标都表现出完美的稳定性（1.0）。

发现*3*： 在不同运行中，多智能体系统调用图在智能体间交互是否发生这一点上总体稳定，但这些交互展开的顺序往往不稳定；因此，可复现性在交互结构层面高于执行顺序层面。

架构决定稳定性，而模型的影响因架构而异。 我们进一步比较不同模型和架构下的执行模式，探究影响调用图相似度的因素。图8展示了跨模型最长公共子序列调用图相似度热图，其中每个子图对应一种架构，每个单元格表示两个模型之间的成对相似度中位数。我们观察到，不同架构呈现出截然不同的稳定性特征：CRAG 的一致性极高（所有模型对的平均相似度为 0.97），而语言智能体树搜索（0.54）和规划与执行（0.47）的变化则显著更大。此外，模型选择的影响取决于架构。在 CRAG 中，大多数模型的调用图完全相同，只有 gemini-2.5-flash-lite 是例外。在语言智能体树搜索中，所有模型生成的调用图都相似。相比之下，在规划与执行中，gpt-4o-mini 与其他模型显著不同，对它们的相似度较低，但其自身不同运行之间仍高度一致；其余模型则往往彼此相似。

发现*4*：多智能体系统架构主导了调用图相似度，而模型选择的影响则因架构而异。

接下来，我们将关注应用层指标，包括成本、任务时长和准确率。由于大语言模型固有的非确定性，准确率可能因运行而异；因此，要确保结果质量，通常需要付出更高的成本并延长执行时间。尽管大语言模型驱动的执行过程具有混乱性，MAESTRO 仍可对这类行为进行系统分析。为确保不同配置之间的比较公平，以下评估仅关注架构套件，以支持更细粒度的分析。

4.4 不同的智能体架构如何影响任务性能和稳定性?

旨在处理各种复杂任务的通用求解器架构，往往会更加谨慎地推进。为了确保稳健性，它们通常会在每次迭代时暂停下来，反思中间状态。例如，规划与执行会先将总体目标分解为一系列里程碑，然后逐步解决每个子任务。这种方法有助于模型全面理解任务上下文，通常也能带来更可靠的结果，但代价是执行时间和资源消耗增加。

相比之下，如果任务类型是预先已知的，就可以采用更具针对性的架构。例如，CRAG 专为基于检索的工作负载而设计。它不会探索其他推理路径，而是优先直接回答查询，尽量减少迂回。这种目标导向的设计力求尽早完成任务，即使背景信息不完整也不例外，并以减少探索换取效率。设计理念上的差异会导致不同架构的执行行为显著不同。

专用求解器可最大限度地降低资源消耗。如图9a所示，在不同的模型选择下，CRAG 始终处于成本更低、延迟更低的区域。具体而言，CRAG 每项任务的中位成本为 $0.0010，比……低一个数量级以上规划与执行（中位成本为 $0.0126）和 LATS（中位成本为 $0.0101）。CRAG 的执行速度也更快，中位任务时长为 42.8 秒，而规划与执行为 101.5 秒。

相比之下，规划与执行的任务时长方差要大得多（四分位距为 30.6–356.6 秒），反映出迭代规划与执行带来的额外开销。LATS 的中位延迟相对较低（32.3 秒），但总体资源成本较高。

准确率会随着架构复杂性的增加而下降。此外，图9b显示，CRAG 的准确率与通用性更强的架构相当，有时甚至更高。CRAG 的平均准确率为 70.6%，而规划与执行为 48.3%，且不同运行间的波动也较小。这些结果表明，针对特定任务设计的智能体架构可以同时降低资源消耗并保持较强的任务性能。

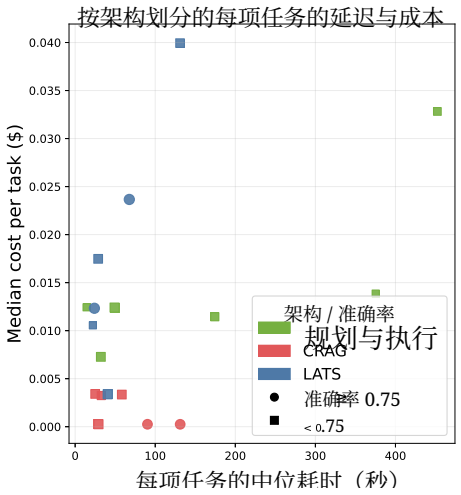
值得注意的是，架构复杂度的提升并不一定会带来更高的准确率，甚至可能适得其反。虽然人们很容易想到加入更多智能体——例如事实核查器或验证阶段——以确保系统表现符合预期，但这类设计不可避免地会增加执行成本并延长交互历史。在我们的评估中，规划与执行在任务推理上耗时显著更长，却取得了更低的准确率（平均为 48.3%，而 CRAG 为 70.6%），同时执行成本也显著更高。这种表现与先前的发现一致：随着交互历史变长，模型性能会下降，因为模型对较早上下文的关注度减弱，且较长推理链中的错误会不断累积 [34]。

发现*5*：更通用的智能体架构会消耗更多资源，且无法稳定提升准确率。

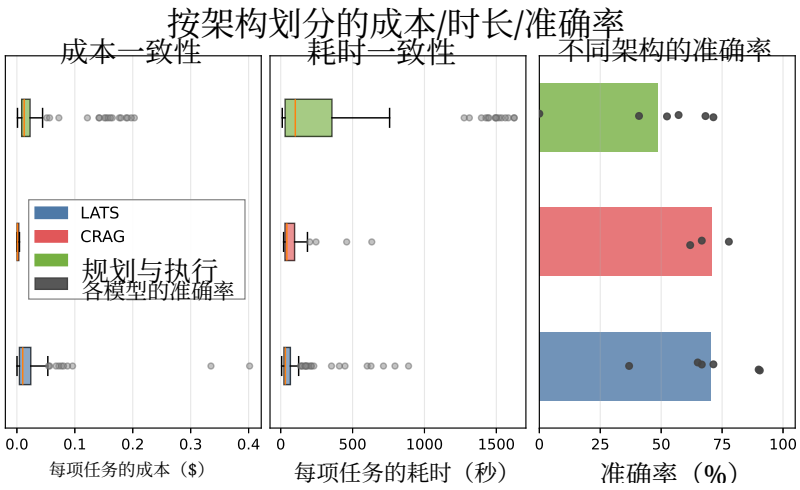
4.5 模型选择如何影响多智能体系统的行为?

基于大语言模型的多智能体系统设计中，一个常见假设是，升级底层模型应能提升系统性能。直观来看，采用能力更强的模型预计会增加成本，同时提高准确率。然而，我们的实验结果对此提出了质疑。我们发现，实际应用中，更强的模型未必会显著增加成本，也不一定能持续提升正确性。相反，模型选择对多智能体系统行为的影响更为微妙，有时甚至出人意料。

更强的模型降低的是迭代开销，而非总成本。能力更强的模型通常能以更少的迭代完成子任务，从而减少反复重试或精炼循环过长等异常行为。然而，这些效率提升主要抵消了较高的每令牌定价，并未转化为更低的总体成本。例如，尽管模型规模不同，gpt-5-mini 和 gpt-5-nano 的每项任务平均成本相近（0.033 对比 0.043），而 gpt-4o-mini 的中位成本（0.0034）则远低于两者。同样，执行延迟并非单调变化：gpt-4o-mini 完成任务的速度比更大的 5系列模型更快（中位数为 45.3 秒），而 gpt-5-nano



(a) **按智能体架构分组的每项任务延迟与成本对比。两个坐标轴上的数值越低，性能越好。**



(b) **不同智能体架构的成本、延迟和准确率分布；专用设计（如 CRAG）能够以较低成本实现稳定的准确率。**

图 9： **不同智能体架构的资源成本与准确率对比。与 LATS 等更通用的架构相比，CRAG 等任务专用设计能以更低的资源成本达到相当的准确率。**

的速度却比名义上规模更小的 gpt-5-mini 慢。这些结果表明，模型选择会影响迭代效率和长尾表现，但不会形成明确的成本层级。

不同模型的准确率呈现非单调且不稳定的趋势。 我们还发现，模型强度与任务准确率之间不存在一贯的关联。虽然 gpt-5-mini 的准确率最高（中位数为 81%），但较弱或价格相近的模型并未呈现可预测的趋势：gpt-5-nano 的准确率仅为 65%；gpt-4o-mini 的准确率中位数较高（71%），但均值明显较低（48%），表明其表现不稳定，且存在大量失败案例。Gemini 各模型的准确率大致相近（约为 66%），其中 2.0-lite 变体的整体表现较差。这些结果表明，多智能体系统的准确率高度依赖执行动态和方差放大，而非仅由模型能力决定；升级基础模型并不足以保证正确率提升。

发现 *6*： 升级基础大语言模型并不能可靠地降低多智能体系统的成本或提高准确率，因为执行动态的影响超过了模型层面的提升。

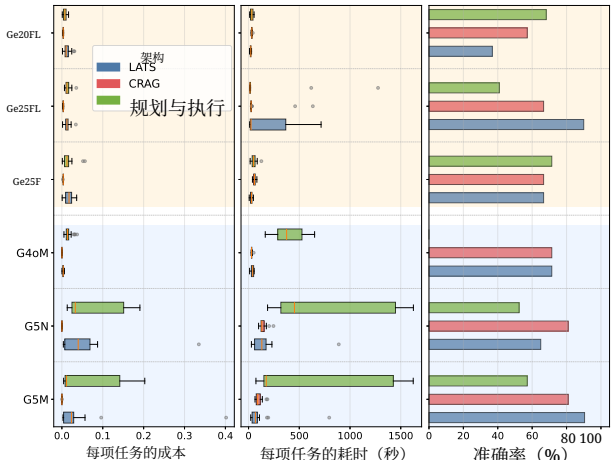


图 10： **大语言模型在成本、时长与准确率方面的权衡；Gemini 系列模型的效率有所提升，而准确率未呈现明确的规模扩展趋势。**

4.6 基于大语言模型的多智能体系统中，主要故障模式有哪些？

我们发现，大多数故障表现为静默灰色错误（75 表 3 中为 17%），这些问题不会触发明确的系统故障，因此用户无法立即察觉。只有人工检查输出时，才会发现这些错误。值得注意的是，这类故障并非系统级异常，而是看似合理却无法使用的响应。因此，基于大语言模型的多智能体系统中的故障归因尤其困难，因为错误的执行往往能够完成，且不会发出任何明确错误信号。

我们还在图 11a 中按模型细分了故障原因，结果显示不同模型具有各自鲜明的故障特征。不同的大语言模型后端并非以相同方式发生故障，而是在出错时表现出各自特有的行为。

特定模型的失效模式。

- **Gemini-2.0-flash-lite**主要通过生成规格不足或不完整的输出而失败：虽然返回了响应，但缺乏足够的细节，无法满足任务要求。

表 3：**所有实验的全局失败构成**。

失败类别	百分比（%）
缺失或说明不充分的输出	47.61
错误事实或实体	27.66
空预测	15.96
异常	6.38
超时	1.86
其他	0.54
静默语义失败	**75.17**
显式失败	24.84

- **Gemini-2.5-flash-lite** 表现出更为保守的失败模式，在不确定时经常选择不作答，并返回空输出或 null。
- **GPT-4o-mini** 往往会给出完整但事实错误的回答，自信地认定错误的实体或事实，而不是省略答案。

这些不同的失败特征表明，多智能体系统的故障不仅取决于模型，还受到智能体架构如何解读和传播部分输出的影响。因此，故障表现为依赖执行路径的现象，而非可归因于单个组件的孤立故障。

发现7：多智能体系统的故障主要表现为隐性的语义错误，具有独特的、因模型而异的失败特征，并会受到执行动态的放大。

大语言模型评判器之间的故障归因存在分歧。为评估基于大语言模型评审的故障归因的可靠性，我们使用另外三个评审模型进行了离线分析，并向每个模型提供多智能体系统的最终回答及对应的标准答案。如图11b所示，由于缺少运行时执行信号，离线归因难以正确识别异常或超时等系统级故障。

例如，多智能体系统的执行可能陷入非终止审查循环，反复生成包含正确答案的回复，却始终无法产生有效的最终输出。在线执行时，这种行为会被正确识别为失败，因为任务并未成功完成。相比之下，离线评判器只能观察最终回复和历史记录，因此可能会错误地将该执行判定为成功，因为轨迹中出现了正确答案。

即使对于基于语义的灰度故障，评审者通常也会就某次执行总体上是正确还是错误达成一致（例如，所有评审者都一致认定 CRAG 执行成功），但在故障类型的归因上仍会出现显著分歧。例如，当 MAS 回复：“很抱歉，我无法回答这个问题。现有工具不具备确定古吉拉特邦立法议会及议会成员所属国家的功能。”时，gpt-oss-120b 评审者将这一结果归类为空预测，而 gemini-2.5-flash 则将其归因于错误的事实/实体。

这些差异表明，即使在输入和故障定义完全相同的情况下，基于大语言模型的评审者也可能在细粒度的故障归因上产生分歧，这凸显了仅离线基于语义的故障分析所固有的主观性和不稳定性。

4.7 工具使用如何影响成本和准确率?

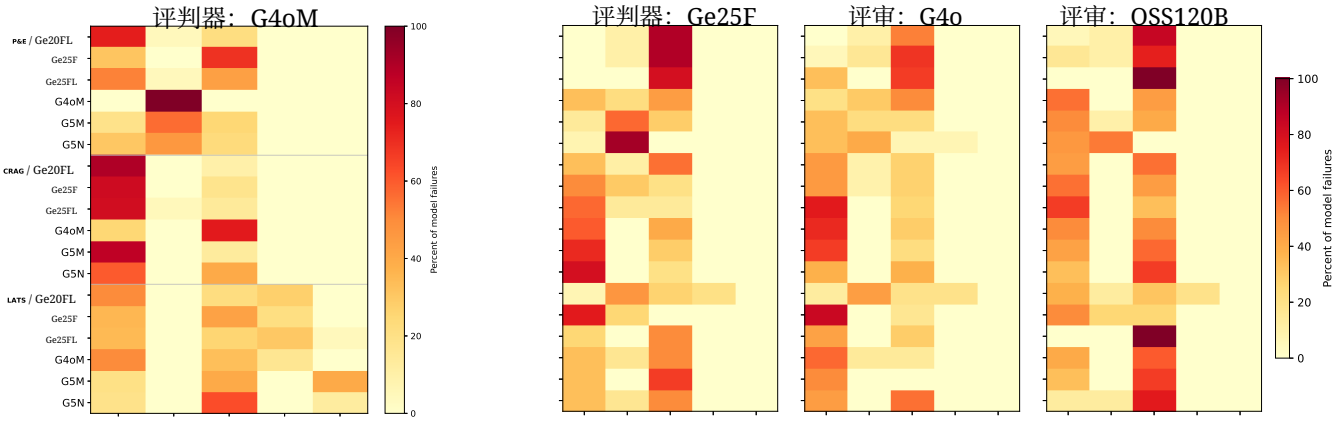
基于大语言模型的多智能体系统设计中，一个常见假设是，启用外部工具应能提升任务性能。通过为智能体提供更多信息来源或能力，人们预期能够获得更高质量的输出，进而提高准确率。然而，我们的结果表明，工具使用的影响高度依赖于底层智能体架构。

启用网络搜索通常会增加资源消耗。总体而言，启用外部工具往往会增加资源消耗，但准确率的提升并非在所有架构中都一致。如图 12a 所示，工具的集成方式不同，其对执行工作流造成的开销也各不相同。对于 CRAG，外部工具主要会增加货币成本，每项任务的中位成本增加 0.0010 美元，中位延迟适度增加 8.1 秒，这反映了额外的检索和处理步骤。相比之下，规划与执行的任务时长显著增加，中位延迟增加 34.1 秒，而货币成本略有下降，表明开销转向了更长的执行时间，而非额外的令牌用量。LATS 的总体开销最高；使用工具会同时增加执行时间和成本，说明交互与协调开销叠加。

当网络搜索缩短任务时长。尽管外部工具通常会带来额外开销，但我们观察到一些显著的例外情况：网络搜索会降低整体执行成本和延迟。具体而言，对于使用 gpt-5-nano 的 CRAG，启用网络搜索后，任务完成速度更快（平均约快 2 秒）。这是因为缺少外部证据时，模型往往会生成更长、更具推测性的回答，从而增加令牌用量和每轮大语言模型延迟。追踪级分析证实了这一现象：在未使用搜索的执行过程中，生成器和评分器会生成更长的输出，大幅增加每次调用的延迟；而提供网络证据后，回答会更简洁，大语言模型延迟也会降低（生成器的中位耗时从 11.2 秒降至 6.1 秒）。因此，尽管增加了检索步骤，使用网络搜索的 CRAG 平均任务时长仍降低了 13.9%，这表明外部上下文可以减少推测性推理，并抵消工具开销。

当网络搜索降低规划成本时。对于规划与执行，启用网络搜索通常会降低总体成本，因为外部证据能让规划器制定更具体、更简洁的规划。没有网络搜索时，规划器往往会生成更长、更具推测性的规划，而重新规划器则会输出更冗长的信息来解释或修订这些规划，从而增加令牌用量。

跟踪级证据支持这一观测：在启用网络搜索时，各模型的规划器消息都明显更短（例如，每次调用的规划器令牌平均用量从 1,500 多降至几百），重新规划器的轮次也更加简洁。尽管规划或重规划迭代的次数可能



（a）**通过大语言模型评审进行在线故障归因（GPT-4o-mini）。**

(b) **通过大语言模型评审进行离线故障归因（Gemini-2.5-Flash、GPT-4o、GPT-OSS-120B）。**

图 11：**使用不同的大语言模型评审进行故障归因。条件允许时，我们优先采用在线归因，因为它能纳入离线时无法获取的运行时信号，例如执行停滞和系统输出不完整。对于离线归因，评审模型会获得多智能体系统的最终响应、相应的标准答案以及相同的故障分类体系。尽管输入受到控制，不同评审模型的归因结果仍存在显著差异，这凸显了基于大语言模型的故障归因固有的主观性和不稳定性。**

保持不变，甚至略有增加，但每轮令牌用量的减少超过了额外网络检索步骤带来的成本，因此整体执行成本更低。

发现 *8*：通过提供外部上下文，工具可以减少推测性生成，从而缩短推理时间并降低成本。

网络搜索可提升准确率，但不同架构受益程度不一。 工具的使用在不同智能体架构中产生了显著不同的结果。如图 12b 所示，CRAG 始终能从外部工具中获益，中位准确率提升 35.7%，且在 83.3% 的评估运行中准确率有所提高。相比之下，规划与执行的中位准确率仅略有下降，且只有三分之一的运行观察到准确率提升。LATS 的提升幅度很小且不稳定，中位准确率提升为 4.2%，仅有一半案例呈现正向效果。

总之，这些准确率趋势与相应的成本和延迟开销相符。CRAG 仅会适度增加成本和执行时间，而规划与执行主要带来更长的执行延迟，LATS 则会同时增加成本和延迟。综合来看，这些结果表明，只有当底层架构能够在不加剧执行复杂性或不稳定性的情况下整合外部工具时，外部工具才能提升多智能体系统的性能。

发现 *9*：只有当智能体架构能够集成外部工具，且不会加剧执行开销或波动时，外部工具才能提升准确率。

5 讨论

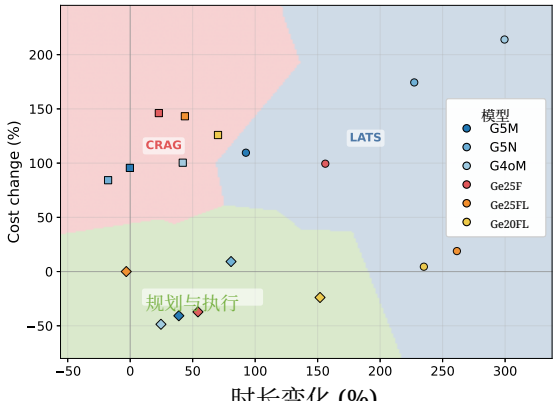
尽管 MAESTRO 已经为基于大语言模型的多智能体系统的行为提供了宝贵洞见，但仍有很大的扩展空间。

5.1 局限性

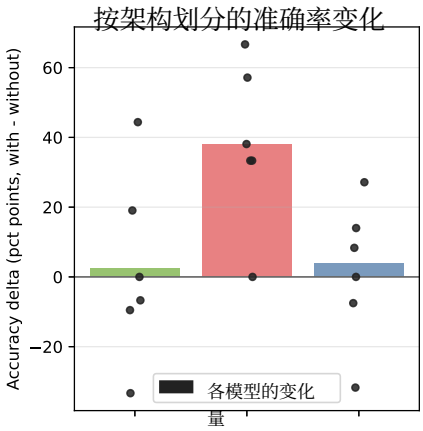
泛化能力。 鉴于基于大语言模型的多智能体系统具有固有的异质性（**D1**），很难确定一种能够适用于所有智能体系统的规范性架构或执行模式。多智能体系统的设计空间仍在快速演进，新的协调策略、工具抽象和执行模型不断涌现。

尽管 MAESTRO 旨在覆盖多种广泛使用的多智能体系统架构和工作流，但我们评估所得的洞见必然以所研究的特定实例和配置为基础。因此，部分发现可能无法直接适用于未来的多智能体系统设计，或未纳入我们基准测试的应用领域。尤其是，智能体编排或模型能力的进步可能会使某些观察结论随时间推移而失效，这凸显了基准测试需要与多智能体系统生态同步持续演进。

遥测开销。 MAESTRO 集成了细粒度遥测，用于检查基于大语言模型的多智能体系统在每个步骤中的行为，并深入洞察执行动态。然而，这类监测埋点会带来性能分析开销，可能降低系统性能。因此，在实际部署之前，MAESTRO 必须优化遥测数据收集，以尽可能降低开销，例如采用采样策略、自适应日志记录或轻量级监控机制。



(a)**启用网页搜索工具的影响。启用网络搜索工具后，我们测量执行延迟和货币成本的变化量。**



(b) **启用网络搜索后的准确率变化量。**

图 12：**不同智能体架构的资源成本与准确率。与 LATS 等更通用的架构相比，CRAG 等针对特定任务设计的架构能够以更低的资源成本实现相当的准确率。**

5.2 未来工作

多智能体系统实例的自动集成。目前，MAESTRO 收录的多智能体系统示例数量有限，可能无法充分展现现实世界部署的多样性与复杂性。为提高代表性，基准测试必须纳入更广泛的多智能体系统示例，以反映现实世界部署和用例的多样性。目前，细粒度遥测是通过针对各个多智能体系统框架定制的临时监测埋点来实现的。作为未来工作，我们计划开发一个自动翻译层，将异构的智能体实现映射为统一的执行表示，从而以最少的人工干预系统地捕获行为。这种自动化集成还有助于降低外部贡献者的参与门槛，让开发者能够更轻松地使用我们的测试套件评估自己的多智能体系统实现。

单体架构与分布式架构。 类似于传统软件架构中单体应用与微服务之间的关系，基于大语言模型的多智能体系统既可以作为单一、统一的系统进行部署，也可以作为通过网络进行通信的分布式智能体集合进行部署。分布式部署能够带来更高的容错性、可扩展性和模块化程度等优势。然而，这也会引入与网络延迟、同步和一致性相关的挑战。未来的工作可以探究单体多智能体系统架构与分布式多智能体系统架构之间的权衡，评估它们在各种工作负载和部署场景下的性能、可靠性与资源利用率。此外，研究底层网络基础设施对分布式基于大语言模型的多智能体系统行为和性能的影响，也将颇具意义。

多智能体系统特有的故障归因。 大语言模型固有的非确定性引入了传统确定性系统中很少遇到的故障模式。当多个智能体被组合成流水线时，这些影响会进一步放大，从而增加出现不一致或涌现性故障行为的可能性。我们的评估已经观察到此类现象。例如，在规划与执行架构中，我们发现了反复出现的执行模式：执行器成功检索并返回标准答案，但重新规划器却一再拒绝中间结果。这种不匹配使系统无法达到终止状态，最终导致超时，尽管执行轨迹中存在正确的解决方案。这些观察结果凸显了基于大语言模型的多智能体系统中故障归因的困难。由于大语言模型异质性（**D1**）所导致的广泛故障空间会随着多个智能体和模型之间的交互呈组合式增长，故障往往无法定位到单个组件或决策点。因此，为此类系统制定有原则的故障分类体系和稳健的归因机制，仍是未来工作的重要方向。

通信机制。 我们的实验揭示了不同多智能体系统框架在实现智能体间通信方面存在显著差异。在我们评估的许多框架中，智能体主要通过结构化函数调用进行交互。另一些框架则依赖共享的全局暂存区，让智能体能够对共同的中间状态进行读写。

对于超出单台主机范围的交互，或访问外部数据源的场景，一些框架还支持标准化的通信协议，例如智能体间（A2A）[12] 消息传递或模型上下文协议（MCP）[13]。这些通信机制上的差异带来了不同的执行语义和协调模式，但它们对系统性能、鲁棒性和失败行为的影响在很大程度上仍未得到探索。这一观察揭示了一个有待研究的领域：了解通信设计选择如何影响基于大语言模型的多智能体系统的行为。

并行性与协调效应。并行性从根本上改变了基于大语言模型的多智能体系统的执行动态，不仅影响吞吐量和资源利用率，也影响协调行为和故障模式。尽管并行执行和负载均衡是传统系统中成熟的技术，但它们在异步多智能体系统中的影响仍未得到充分理解。

现有研究已广泛探讨单个大语言模型的优化方法，例如推测推理和并行解码 [6, 30], ，以提高准确率或改善成本–性能权衡。然而，这些技术如何应用于多智能体场景尚不明确；在这类场景中，多个智能体可能并发运行，并通过共享状态或工具进行交互。

尤其是，具有重叠或部分冗余角色的并行智能体所产生的影响尚未得到充分刻画。这类配置可能会引入新的协调开销、争用或涌现行为，与单模型并行有根本性差异，这也凸显了未来研究的一个重要方向。

框架开销研究。在我们的评估中，我们观察到，kagent [46], ——一个旨在促进构建分布式大语言模型多智能体系统的框架——可能会产生不容忽视的通信开销，也可能触发运行故障（例如，kagent控制器节点发生磁盘已满错误）。未来的工作应系统地刻画多智能体系统框架引入的开销，并量化其对端到端性能和可靠性的影响。

6 结论

我们认为，对基于大语言模型的多智能体系统（MAS）的评估，不应仅看任务完成情况，还应将其视为具有动态、随机执行特征的复杂系统。为此，我们推出了 MAESTRO，这是一个开源评估套件，可标准化异构多智能体系统的配置与执行，同时导出细粒度的系统级遥测数据，以支持跨栈比较。

我们对 12 个具有代表性的多智能体系统实例进行评估后发现，尽管智能体工作流通常在结构上 较为稳定，但在时间上 却表现出显著的不稳定性，导致延迟、成本和故障模式的运行间方差较大。尤为重要的是，我们发现，在决定资源配置、可复现性以及成本—延迟—准确率权衡方面，多智能体系统架构的影响超过后端模型和工具集选择。这些发现表明，提升智能体系统的可靠性和效率，根本上是一个架构层面的挑战，因此需要优先关注深度执行可见性，而非简单应用层评分的基准测试。

展望未来，我们计划扩展 MAESTRO，以支持分布式架构和智能体自动集成，同时改进故障归因，以便更准确地诊断随机错误。我们的最终目标是建立标准化的可观测性契约，确保基准测试能够跟上智能体系统不断演进的复杂性。

参考文献

[1] AutoGen。2025年。AutoGenBench。https://github.com/microsoft/autogen/tree/main/ python/packages/agbench 访问日期：2025年12月28日。[2] Kinjal Basu、Ibrahim Abdelaziz、Kiran Kate、Mayank Agarwal、Maxwell Crouse、Yara Rizk、Kelsey Bradford、Asim Munawar、Sadhana Kumaravel、Saurabh Goyal 等。2025年。Nestful：用于评估大语言模型嵌套API调用序列的基准测试。载于自然语言处理实证方法会议（EMNLP）。33526–33535。[3] BIG bench作者。2023年。超越模仿游戏：量化并外推语言模型的能力。机器学习研究汇刊（TMLR）（2023年）。https://openreview.net/forum?id=uyTL5Bvosj [4] Tara Bogavelli、Roshnee Sharma 和 Hari Subramani。2025年。AgentArch：用于评估企业中智能体架构的综合基准测试。arXiv预印本 arXiv:2509.10769（2025年）。[5] Mert Cemri、Melissa Z Pan、Shuyi Yang、Lakshya A Agrawal、Bhavya Chopra、Rishabh Tiwari、Kurt Keutzer、Aditya Parameswaran、Dan Klein、Kannan Ramchandran 等。2025年。多智能体大语言模型系统为何失败？ arXiv预印本 arXiv:2503.13657（2025年）。[6] Charlie Chen、Sebastian Borgeaud、Geoffrey Irving、Jean-Baptiste Lespiau、Laurent Sifre 和 John Jumper。2023年。利用推测采样加速大语言模型解码。arXiv预印本 arXiv:2302.01318（2023年）。[7] Krishna Teja Chitty-Venkata、Siddhisanket Raskar、Bharat Kale、Farah Ferdaus、Aditya Tanikanti、Ken Raffenetti、Valerie Taylor、Murali Emani 和 Venkatram Vishwanath。2024年。Llm-inference-bench：人工智能加速器上的大语言模型推理基准测试。载于国际高性能计算、网络、存储与分析会议研讨会。IEEE，1362–1379。[8] Comet ML。2024年。Opik：开源大语言模型评估与可观测性平台。https://github.com/comet-ml/opik 访问日期：2025年12月14日。[9] Alessandro Cornacchia、Vaastav Anand、Muhammad Bilal、Zafar Qazi 和 Marco Canini。2025年。DMAS-Forge：将人工智能应用透明地部署为分布式系统的框架。智能体人工智能系统研讨会（SAA）（2025年）。[10] 微软公司。2024年。AutoGen。https://microsoft.github.io/autogen/stable//index.html 访问日期：2025年12月28日。[11] Darshan Deshpande、Varun Gangal、Hersh Mehta、Jitin Krishnan、Anand Kannappan 和 Rebecca Qian。2025年。TRAIL：轨迹推理与智能体问题定位。arXiv预印本 arXiv:2505.08638（2025年）。[12] A2A开发者。2025年。A2A协议规范。https://a2a-protocol.org/latest/topics/what-is-a2a/ 访问日期：2025年12月9日。[13] MCP开发者。2025年。模型上下文协议规范。https:// modelcontextprotocol.io/docs/getting-started/intro 访问日期：2025年12月9日。[14] Adam Fourney、Gagan Bansal、Hussein Mozannar、Cheng Tan、Eduardo Salinas、Erkang (Eric) Zhu、Friederike Niedtner、Grace Proebsting、Griffin Bassman、Jack Gerrits、Jacob Alber、Peter Chang、Ricky Loynd、Robert West、Victor Dibia、Ahmed Awadallah、Ece Kamar、Rafah Hosn 和 Saleema Amershi。2024年。Magentic-One：用于解决复杂任务的通用多智能体系统。技术报告 MSR-TR-2024-47。Microsoft。https://www.microsoft.com/en-us/research/publication/magentic-one-a-generalist-multi-agent-system-for-solving-complex-tasks/ [15] Longling Geng 和 Edward Y Chang。2025年。Realm-bench：面向大语言模型和多智能体系统的真实场景规划基准测试。arXiv预印本 arXiv:2502.18836（2025年）。[16] 谷歌。2025年。Google ADK示例代码库。https://github.com/google/adk-samples/tree/main 访问日期：2025年12月28日。[17] 谷歌。2025年。Google智能体开发套件（ADK）。https://google.github.io/adk- docs/ 访问日期：2025年12月28日。[18] 谷歌。2025年。Google GenAI搜索工具。https://googleapis.github.io/python-genai/genai.html#genai.types.GoogleSearch 访问日期：2025年12月28日。[19] 谷歌。2025年。Google Imagen 3模型。https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/imagen/3-0-generate 访问日期：2025年12月28日。[20] Taicheng Guo、Xiuying Chen、Yaqi Wang、Ruidi Chang、Shichao Pei、Nitesh V. Chawla、Olaf Wiest 和 Xiangliang Zhang。2024年。基于大语言模型的多智能体：进展与挑战综述。载于国际人工智能联合会议（IJCAI），Kate Larson（编）。8048–8057。doi:10.24963/ijcai.2024/890 Survey Track。[21] Taicheng Guo、Xiuying Chen、Yaqi Wang、Ruidi Chang、Shichao Pei、Nitesh V Chawla、Olaf Wiest 和 Xiangliang Zhang。2024年。基于大语言模型的多智能体：进展与挑战综述。arXiv预印本 arXiv:2402.01680（2024年）。[22] Zhicheng Guo、Sijie Cheng、Hao Wang、Shihao Liang、Yujia Qin、Peng Li、Zhiyuan Liu、Maosong Sun 和 Yang Liu。2024年。StableToolBench：面向大语言模型工具学习的大规模稳定基准测试。arXiv预印本 arXiv:2403.07714（2024年）。[23] Dan Hendrycks、Collin Burns、Steven Basart、Andy Zou、Mantas Mazeika、Dawn Song 和 Jacob Steinhardt。2020年。大规模多任务语言理解测量。arXiv预印本 arXiv:2009.03300（2020年）。[24] CrewAI Inc. 2024年。CrewAI。https://www.crewai.com 访问日期：2025年12月28日。

[25] LangChain 公司。2024年。LangGraph。https://langchain-ai.github.io/langgraph/ 访问日期：2025年12月28日。[26] Gurusha Juneja、Jayanth Naga Sai Pasupulati、Alon Albalak、Wenyue Hua 和 William Yang Wang。2025年。MAGPIE：多智能体上下文隐私评估基准测试。arXiv预印本 arXiv:2510.15186（2025年）。[27] Fan Lai、Yinwei Dai、Sanjay Singapuram、Jiachen Liu、Xiangfeng Zhu、Harsha Madhyastha 和 Mosharaf Chowdhury。2022年。FedScale：大规模联邦学习的模型与系统性能基准测试。载于国际机器学习会议（ICML）。PMLR，11814–11827。[28] LangChain文档。2025年。ChatVertexAI — LangChain Vertex AI参考。https://reference.langchain.com/python/integrations/langchain_google_vertexai/ChatVertexAI/ 访问日期：2025年12月28日。[29] LastMile AI。2024年。MCP 财务分析器：多智能体 MCP 应用示例。访问日期：2025年12月31日。[30] Yaniv Leviathan、Matan Kalman 和 Yossi Matias。2023年。通过推测解码实现基于Transformer模型的快速推理。载于国际机器学习会议（ICML）。PMLR，19274–19286。[31] Xuechen Li、Tianyi Zhang、Yann Dubois、Rohan Taori、Ishaan Gulrajani、Carlos Guestrin、Percy Liang 和 Tatsunori B. Hashimoto。2023年。AlpacaEval：指令遵循模型的自动评估器。https://github.com/tatsu-lab/alpaca_eval。[32] Percy Liang、Rishi Bommasani、Tony Lee、Dimitris Tsipras、Dilara Soylu、Michihiro Yasunaga、Yian Zhang、Deepak Narayanan、Yuhuai Wu、Ananya Kumar、Benjamin Newman、Binhang Yuan、Bobby Yan、Ce Zhang、Christian Alexander Cosgrove、Christopher D Manning、Christopher Re、Diana Acosta-Navas、Drew Arad Hudson、Eric Zelikman、Esin Durmus、Faisal Ladhak、Frieda Rong、Hongyu Ren、Huaxiu Yao、Jue WANG、Keshav Santhanam、Laurel Orr、Lucia Zheng、Mert Yuksekgonul、Mirac Suzgun、Nathan Kim、Neel Guha、Niladri S. Chatterji、Omar Khattab、Peter Henderson、Qian Huang、Ryan Andrew Chi、Sang Michael Xie、Shibani Santurkar、Surya Ganguli、Tatsunori Hashimoto、Thomas Icard、Tianyi Zhang、Vishrav Chaudhary、William Wang、Xuechen Li、Yifan Mai、Yuhui Zhang 和 Yuta Koreeda。2023年。语言模型的整体评估。机器学习研究汇刊（TMLR）（2023年）。https://openreview.net/forum?id=iO4LZibEqW 精选认证、专家认证。[33] Wang Lilian。2023年。由大语言模型驱动的自主智能体。https://lilianweng.github.io/posts/2023-06-23-agent/ 访问日期：2025年12月28日。[34] Nelson F Liu、Kevin Lin、John Hewitt、Ashwin Paranjape、Michele Bevilacqua、Fabio Petroni 和 Percy Liang。2024年。迷失于中间：语言模型如何使用长上下文。计算语言学协会汇刊（TACL）12（2024年），157–173。[35] Xiao Liu、Hao Yu、Hanchen Zhang、Yifan Xu、Xuanyu Lei、Hanyu Lai、Yu Gu、Hangliang Ding、Kaiwen Men、Kejuan Yang、Shudan Zhang、Xiang Deng、Aohan Zeng、Zhengxiao Du、Chenhui Zhang、Sheng Shen、Tianjun Zhang、Yu Su、Huan Sun、Minlie Huang、Yuxiao Dong 和 Jie Tang。2023年。AgentBench：将大语言模型作为智能体进行评估。arXiv预印本 arXiv:2308.03688（2023年）。[36] LlamaIndex。2024年。LlamaIndex。https://www.llamaindex.ai 访问日期：2025年12月28日。[37] Agno维护者。2025年。Agno 团队。https://docs.agno.com/concepts/teams/ introduction 访问日期：2025年12月28日。[38] Microsoft。2025年。AutoGen AgentChat用户指南。https://microsoft.github.io/ autogen/stable//user-guide/agentchat-user-guide/ 访问日期：2025年12月28日。[39] Dany Moshkovich、Hadar Mulian、Sergey Zeltyn、Natti Eder、Inna Skarbovsky 和 Roy Abitbol。2025年。超越黑盒基准测试：智能体系统的可观测性、分析与优化。arXiv预印本 arXiv:2503.06745（2025年）。[40] OpenTelemetry作者。2025年。OpenTelemetry。https://opentelemetry.io/ 访问日期：2025年12月28日。[41] Melissa Z Pan、Negar Arabzadeh、Riccardo Cogo、Yuxuan Zhu、Alexander Xiong、Lakshya A Agrawal、Huanzhi Mao、Emma Shen、Sid Pallerla、Liana Patel 等。2025年。在生产环境中测量智能体。arXiv预印本 arXiv:2512.04123（2025年）。[42] A2A 项目。2025年。A2A示例代码库。https://github.com/a2aproject/a2a- samples 访问日期：2025年12月28日。[43] Psutil开发者。2025年。Psutil：进程和系统实用工具。https://github.com/ giampaolo/psutil 访问日期：2025年12月28日。[44] Bronson Schoen、Evgenia Nitishinskaya、Mikita Balesni、Axel Højmark、Felix Hofstätter、Jérémy Scheurer、Alexander Meinke、Jason Wolfe、Teun van der Weij、Alex Lloyd 等。2025年。对审慎对齐进行压力测试以开展反谋划训练。arXiv预印本 arXiv:2509.15541（2025年）。[45] Noah Shinn、Federico Cassano、Ashwin Gopinath、Karthik Narasimhan 和 Shunyu Yao。2023年。Reflexion：基于语言强化学习的语言智能体。神经信息处理系统进展（NeurIPS）36（2023年），8634–8652。[46] Solo.io。2025年。kagent：云原生智能体人工智能框架。https://kagent.dev/ 访问日期：2025年12月28日。[47] Haochen Sun、Shuwen Zhang、Lujie Niu、Lei Ren、Hao Xu、Hao Fu、Fangkun Zhao、Caixia Yuan 和 Xiaojie Wang。2025年。Collab-Overcooked：将大语言模型作为协作智能体进行基准测试与评估。arXiv预印本 arXiv:2502.20073（2025年）。

[48] Lei Wang、Wanyu Xu、Yihuai Lan、Zhiqiang Hu、Yunshi Lan、Roy Ka-Wei Lee 和 Ee-Peng Lim。2023年。计划并解决提示：通过大语言模型改进零样本思维链推理。*arXiv* 预印本 *arXiv:2305.04091*（2023年）。[49] Wei Wang、Dan Zhang、Tao Feng、Boyan Wang 和 Jie Tang。2024年。Battleagent-bench：用于评估多智能体系统中语言模型合作与竞争能力的基准测试。*arXiv* 预印本 *arXiv:2408.15971*（2024年）。[50] Yuxin Wang、Yuhan Chen、Zeyu Li、Xueze Kang、Yuchu Fang、Yeju Zhou、Yang Zheng、Zhenheng Tang、Xin He、Rui Guo 等。2025年。Burstgpt：用于优化大语言模型服务系统的真实世界工作负载数据集。载于*ACM*知识发现与数据挖掘会议（*SIGKDD*）。5831–5841。[51] Qingyun Wu、Gagan Bansal、Jieyu Zhang、Yiran Wu、Beibin Li、Erkang Zhu、Li Jiang、Xiaoyun Zhang、Shaokun Zhang、Jiale Liu 等。2024年。Autogen：通过多智能体对话赋能新一代大语言模型应用。载于语言建模会议（*COLM*）。[52] Binfeng Xu、Zhiyuan Peng、Bowen Lei、Subhabrata Mukherjee、Yuchen Liu 和 Dongkuan Xu。2023年。Rewoo：将推理与观察解耦，以实现高效的增强型语言模型。*arXiv* 预印本 *arXiv:2305.18323*（2023年）。[53] Bingyu Yan、Zhibo Zhou、Litian Zhang、Lian Zhang、Ziyi Zhou、Dezhuang Miao、Zhoujun Li、Chaozhuo Li 和 Xiaoming Zhang。2025年。超越自我对话：以通信为中心的基于大语言模型的多智能体系统综述。*arXiv* 预印本 *arXiv:2502.14321*（2025年）。[54] Shi-Qi Yan、Jia-Chen Gu、Yun Zhu 和 Zhen-Hua Ling。2024年。纠正式检索增强生成。（2024年）。[55] Zhilin Yang、Peng Qi、Saizheng Zhang、Yoshua Bengio、William W. Cohen、Ruslan Salakhutdinov 和 Christopher D. Manning。2018年。HotpotQA：用于多样化、可解释的多跳问答的数据集。载于自然语言处理实证方法会议（*EMNLP*）。[56] Shunyu Yao、Noah Shinn、Pedram Razavi 和 Karthik Narasimhan。2024年。Tau-Bench：用于真实世界领域中工具—智能体—用户交互的基准测试。*arXiv* 预印本 *arXiv:2406.12045*（2024年）。[57] Shunyu Yao、Dian Yu、Jeffrey Zhao、Izhak Shafran、Tom Griffiths、Yuan Cao 和 Karthik Narasimhan。2023年。思维树：利用大语言模型进行审慎问题求解。神经信息处理系统进展（*NeurIPS*）36（2023年），11809–11822。[58] Shunyu Yao、Jeffrey Zhao、Dian Yu、Nan Du、Izhak Shafran、Karthik R Narasimhan 和 Yuan Cao。2022年。React：在语言模型中协同推理与行动。载于国际学习表征会议（*ICLR*）。[59] Andy Zhou、Kai Yan、Michal Shlapentokh-Rothman、Haohan Wang 和 Yu-Xiong Wang。2023年。语言智能体树搜索统一了语言模型中的推理、行动与规划。*arXiv* 预印本 *arXiv:2310.04406*（2023年）。

A 附录

A.1 后处理组件详情

我们构建了一个观测组件，用于刻画多智能体系统在不同基准任务和配置下的行为。默认情况下，我们会为每种工作负载生成一组通用图表。我们既报告单次执行的结果，也报告多次运行的汇总结果。由于图表总量庞大，我们计划将其作为数据集发布，并在此仅提供简要总结。

- **令牌消耗**。对于单次运行，我们绘制每个智能体随时间变化的令牌消耗，包括提示令牌和补全令牌；还绘制所有智能体随时间变化的令牌总消耗。对于多次运行，我们绘制所有智能体随时间变化的平均令牌总消耗。

- **延迟**。对于单次运行，我们绘制整个系统的端到端延迟，并将其分解为智能体处理延迟、智能体与大语言模型通信延迟以及智能体之间的通信延迟。对于多次运行，我们绘制整个系统的平均端到端延迟，并展示平均延迟在不同组成部分上的细分情况。此外，我们还绘制延迟火焰图。• **CPU 和内存使用情况**。对于单次运行，我们绘制整个系统的 CPU 和内存使用情况时间序列，并分析 CPU/内存使用情况与系统事件（例如智能体调用、大语言模型调用等）之间的相关性。对于多次运行，我们绘制整个系统 CPU 和内存使用情况的均值、峰值和最小值。• **消息大小**。对于单次运行，我们绘制每个智能体在智能体间消息和智能体与大语言模型之间的消息中的平均输入和输出消息大小。对于多次运行，我们绘制智能体间消息以及智能体与大语言模型之间消息的输入和输出消息总大小。此外，我们还绘制每个智能体、每对智能体以及每个智能体与大语言模型之间的消息大小。• **调用图**。我们将系统中智能体的调用图可视化，并使用图相似度指标展示不同运行的调用图之间的相似度。我们采用两种相似度指标：杰卡德相似度和最大公共子图（LCS）相似度。杰卡德相似度衡量调用图中两组边之间的相似度，而 LCS 相似度衡量调用图中边的序列相似度。

A.2 采集器实现。

A.2.1 遥测字段收集。 以下代码清单（清单 1）展示了 MAESTRO 中遥测字段的收集。

清单 1：遥测字段收集

```
[
  { " trace_id ": "<32-十六进制-跟踪-ID >", " span_id ": "<16-十
" 开始_时间 ": 0, " end_time ": 0, " duration_ns ": 0, "kind ": "<内部
```

```
        " 智能体. 输出. 无用 ": false ,
        " 智能体 . 输出 . 无用_原因 ":                "<自由 -文本 >",
        " 通信. 输入_消息_大小_字节 ": 0,
        " 通信。输出_消息_大小_字节 ": 0,
        " 通信。总计_消息_大小_字节 ": 0
      },
      "通信 ": {
        "在_进程_调用中_吗 ": false          ,
        "输入_消息_大小_字节 ": 0,
        "输出_消息_大小_字节 ": 0,
        "总计_消息_大小_字节 ": 0
      },
      "资源 ": {
        "属性 ": {
          " 服务 . 名称 ":          "<服务 -名称 >"        ,
          " 服务. 版本 ": "<语义化版本 >",
          " 部署 . 环境 ": "< 本地 |开发 | 预发布环境 |生产环境 >",
          " 遥测 .软件开发工具包. 名称 ":   "<软件开发工具包 -名称 >",
          " 遥测 .软件开发工具包. 语言 ":     "<语言 >",
          " telemetry .sdk. version ":       "<version >",
          " host . name ":   "<可选 -主机 >"
        }
      }
    }
]
```

A.2.2 缺乏标准化的可观测性契约。即使采用了通用的可观测性模式（例如 OTEL），不同的编排技术栈在呈现、转换或抑制哪些遥测信号方面仍存在显著差异。一些框架会将令牌用量、终止原因或负载大小等执行元数据传递给应用级钩子，而另一些则在内部执行层中处理这些信号，不向外部公开。因此，相同的智能体工作流可能会因模型后端、传输机制和编排框架的组合不同，而呈现出显著不同的可观测性特征。

造成这种差异的一个关键原因是，与生成文本不同，令牌用量并未被视为具有明确定义暴露契约的一等执行产物，而是被视为辅助元数据。因此，令牌用量是否可观测，取决于以下因素的共同作用：(i) 底层模型 API 及其响应模式；(ii) 传递推理结果所经过的传输层（例如流式传输与非流式传输）；以及 (iii) 框架的检测机制和日志传播策略。在缺乏一致认可的契约时，每一层都会独立决定如何表示令牌用量，以及是否将其转发出去，从而使端到端可观测性变得脆弱，并依赖于具体技术栈。

使用量元数据因后端和调用方式而异，可能会丢失。这种缺乏标准化的情况既存在于不同模型后端之间，也存在于不同调用方式之间。例如，Gemini 和 Vertex AI 确实会提供令牌用量信息，但其响应格式和术语与 OpenAI 或 Anthropic 风格的 API 不同。令牌用量可能通过特定后端的元数据字段报告（例如，将生成端令牌使用量报告为候选令牌，而非 OpenAI 风格的输出令牌或补全令牌 [28]），因此需要借助后端感知解析逻辑才能提取用量信息。除生成 API 外，我们还发现，对于非生成类调用，使用量元数据可能会在框架层被完全丢弃。在 LangGraph 中，嵌入模型调用（例如 OpenAIEmbeddings 和 VertexAIEmbeddings）不会传递令牌用量信息，即使底层提供商 API 支持使用量核算也是如此。在这种情况下，框架会在内部使用部分响应元数据，却不将其转发到应用层遥测或核算钩子。因此，LangGraph 和 MCP-Agent 等编排框架——其中许多默认假定使用同步的 OpenAI 风格使用量模式——除非实施明确的、能感知后端和调用方式的监测埋点，否则可能无法捕获各种执行路径中的令牌用量。重要的是，这些局限并非源于智能体逻辑或后端信号缺失，而是因为缺少稳定的跨提供商可观测性契约，来规定使用量元数据应如何构建、保留并跨越抽象边界传递。

对多智能体系统基准测试的影响。 这种不一致造成了成本和效率分析中的盲点，尤其是在不同大语言模型后端共存的异构多智能体环境中。这进一步表明，不能假定可观测性属性与模型无关，因此有必要制定标准化的可观测性契约，明确规定大语言模型 API 和智能体框架必须暴露哪些执行信号。