软件定义的智能体服务

Saurabh Agarwal MarcoLaju JayanthSrinivasa德克萨斯大学奥斯汀分校 德克萨斯大学
奥斯汀分校 Cisco-Research
Myungjin
LeeCisco-Research
Aditya Akella德克
萨斯大学奥斯汀分校

arXiv:2601.03197v1 [cs.DC] 6 Jan 2026

摘要

随着多智能体大语言模型流水线日益复杂，现有服务范式无法适应动态的服务条件。我们认为，智能体服务系统应具备可编程性并能感知系统状态，而不同于现有服务方式对参数进行静态编码。在这项工作中，我们提出了一种受软件定义网络（SDN）启发的新型智能体服务框架，可根据运行时状态控制通信的关键属性。这种架构能够实现高效、响应迅速的智能体系统服务，并为高层意图驱动的智能体服务铺平道路。

1 引言

大语言模型（LLM）的兴起催生了不断壮大的智能大语言模型驱动的智能体生态系统——每个智能体都作为专门的模型，执行代码生成、测试、检索和规划等任务。为了完成这些复杂任务，这些智能体需要定期调用API、工具和其他模型，形成“智能体流水线” [11, 14]。

以图1所示的智能体软件开发流水线为例。该工作流涉及两个智能体之间的通信：一个负责输出函数的“开发者”智能体，以及一个提供反馈的“测试”智能体。智能体之间有多种可能的通信策略：(a)批处理 一次调用中处理所有函数，这可以减少开销并提高吞吐量。(b)逐函数 流水线处理，从而实现早期反馈和并发。(c) 令牌级流式传输，可将延迟降至最低，以实现交互式响应。在此示例中，“最佳”策略不仅取决于应用程序需求，也取决于当前系统状态。例如，在高负载下，批处理可以保持吞吐量并减少积压；在低负载下，流式传输可以提升应用程序响应能力。更为重要的是，智能体服务决策——包括批大小、路由和通信粒度——既会影响运行时系统因素，也会受到这些因素的影响，例如负载、队列长度、模型延迟、令牌生成速率、用户响应目标、服务级别目标等（§2）。因此，理想情况下，这些决策应当动态地适应服务系统状态。

遗憾的是，如今的智能体服务架构难以选出最优策略，尤其是在系统动态变化时。这是因为现有智能体服务设计存在三个根本性缺陷。首先，如今的智能体通信协议，例如 Anthropic 的模型上下文协议（MCP） [3], Google 的智能体到智能体（A2A） [4],IBM 的智能体连接协议（ACP） [6], 以及其他协议 [2],，要求开发人员在设计时就提前绑定到某种策略上——例如选择批处理、流式传输或流水线发送。其次，批大小、通信频率、路由策略和模型选择等服务参数并未考虑服务层参数，例如模型延迟、排队时间、单次推理成本或跨智能体资源争用，而性能在很大程度上取决于这些因素。最后，如今的通信架构缺少用于跨智能体和全流水线优化的接口，例如对下游智能体进行优先级排序，或在用户体验与端到端系统吞吐量之间取得平衡。

总而言之，这些特性使智能体服务架构既成为智能体流水线开发人员的负担——难以确保和控制性能——也是运行时严重低效的根源。

我们主张采用一种新的智能体服务架构，使智能体通信能够动态控制，并与服务系统紧密耦合。具体而言，受软件定义网络（SDN）的启发，我们主张构建可编程智能体基底。然而，与软件定义网络不同，我们要求该基底不只支持通信，还要直接与应用程序对接，以便更好地控制服务，确保效率和性能。

我们设想的软件定义智能体服务栈包含三个组件：

- 一种**数据平面**，支持细粒度、可动态调整的智能体消息粒度，范围涵盖令牌级流式传输到批处理上下文。
- 一种**指标平面**，通过简单的跨智能体/工具统一数据采集接口，以低开销访问指标。
- 一个**控制平面**利用指标动态管理智能体流水线中的智能体间通信策略、批大小、路由和模型选择。它会系统性地演进系统级状态，使其达到满足运营者指定目标的状态。控制决策会综合考虑系统级和应用程序级指标。

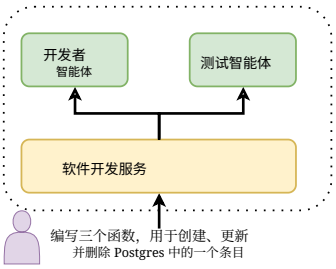


图1：智能体软件开发者：上图展示了一种软件智能体工作流，其中开发者智能体负责生成函数，而测试智能体则生成测试代码。

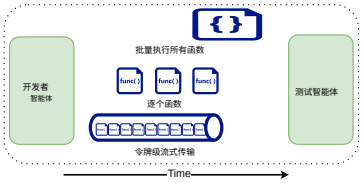


图 2：通信模式：上述示意图给出了另一种可能通信模式的大致时间线。

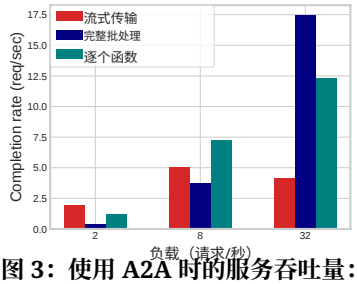


图 3：使用 A2A 时的服务吞吐量：我们在不同负载下，使用三种不同的通信机制为两个智能体提供服务。没有一种配置始终优于其他配置。

我们设想，这样的技术栈能够支持用于智能体通信和服务策略的高级声明式语言。这些语言可让基础设施工程师表达目标，例如“尽可能降低端到端延迟”“除非负载较低，否则进行批处理”或“仅对交互式或高优先级任务使用流处理”，而无需管理底层实现细节。控制平面会根据当前服务状态，将这些意图转化为实时决策。该架构使智能体工作流开发者不必再自行实现控制应用程序性能的定制方案；相反，智能体流水线只需向控制平面提供适当的提示即可。

我们探讨了在实现上述愿景时，工具和智能体的异构性与动态变化所带来的挑战。我们提出了简洁的统一接口，以及面向三个平面的设计思路，以便即使存在这些特性也能实现有效控制。

我们的初步原型表明，通过对通信粒度进行细粒度控制，我们的架构可以将智能体服务吞吐量提升至多达 3.6× 。此外，对服务系统施加额外控制后，服务吞吐量还可进一步提升 2.3×。

与其说我们旨在提出一项具体方案，不如说我们的主要目标是开启一场讨论，探讨如何解决新兴智能体架构中的根本问题。我们提出了一个受 SDN 启发的框架，以及可与当前和未来的工具及智能体集成的应用程序接口，以克服这些问题。但我们也承认，这些只是这一有趣设计空间中的起点。最后，我们在 §5 中指出了未来的关键研究方向。

2 智能体服务

大语言模型正迅速超越聊天机器人的基础用途，日益成为功能全面的数字助理。尤其是，大语言模型正获得使用更多工具的权限 [12,13, 17] （例如计算器、网络搜索引擎、数据库、文件

```python
# 客户端 初始化与 能力 确定
async with httpx.client() as httpx_client:
    developer_agent = A2AClient.get_client_from_agent_card_url(
    httpx_client, "http://本地主机:6069"
)

    测试代理 =A2AClient.get_client_from_agent_card_url(
    httpx_client, "http://本地主机:6070"
)
    提示词 = "编写␣三个␣函数␣来␣创建␣一个␣条目，
␣␣␣␣␣更新␣它␣并␣删除␣它"

    send_message_payload =message{
        'role': "用户" ,
        '部分': [{'类型': '文本','文本': 提示}],
    }

    流式传输_请求 = SendStreamingMessageRequest(
                参数=MessageSendParams(**send_message_payload)
            )
        # 执行流式消息
    流式响应 =developer_agent.send_message_streaming(
        流式请求)
        # 等待流式响应
    异步 遍历 块，位于 流式响应:
        # 向测试智能体进行流处理
          测试请求 =SendStreamingMessageRequest(
                参数=MessageSendParams(**chunk)
        )
        测试函数 =testing_agent.send_message_streaming(chunk)
        await asyncio.sleep(0.1)
```

图 4：使用 A2A 编写的智能体应用程序代码片段示例。

系统等）。此外，它们也被视为自主智能体，与其他智能体协作完成复杂任务（例如，软件开发 [5]和问题求解 [15]）。此类工作流要求大语言模型与其他工具和大语言模型进行通信。

遗憾的是，如今的智能体服务系统存在根本性局限。我们先来介绍这些系统的工作方式。

2.1 智能体工作流

如今，构建多智能体大语言模型工作流需要开发人员手动编排工具和模型之间的通信。开发人员需要整理可调用智能体列表、处理身份验证，并管理

每个端点的通信模式——通常使用 REST、RPC 或流式 API 等异构机制。为管理这种复杂性，近期的工作引入了基于协议的抽象。例如 Google 的 A2A [4], Anthropic 的 MCP [3], 和 IBM 的 ACP [6]。这些协议为智能体发现和消息交换提供通用接口，旨在实现智能体工作流的标准化。图 4 展示了使用 A2A 连接“开发者”智能体与“测试”智能体的示例。该工作流会获取智能体卡片（第 3 行和第 7 行）、向开发者智能体发送流式消息（第 21 行），并将响应转发给测试者智能体（第 24 行）。基础设施工程师负责部署这些工作流，并手动调整通信方式、批大小、路由方案及其他参数，以实现预期性能。

2.2 当前的局限

早期绑定。 虽然 A2A 等协议简化了智能体连接，但要求提前绑定 通信策略。开发人员必须在设计时决定智能体是通过批处理、流式传输还是同步调用进行交互。这些决策（例如图 4 中的第 22 行和第 28 行）被硬编码到应用程序中，尽管理想的通信模式可能会随运行时条件而变化。如图 3 所示，次优策略可能导致性能降低多达 3.6×。

缺乏全局可见性。现有的智能体服务系统没有提供任何机制，使智能体通信能够了解全局服务系统状态。例如，假设某个智能体有两个实例，其中一个实例出现了严重的队列积压。目前，无法将此类队列积压的情况告知工作流中的其他上游智能体，以便进行重新路由或更改通信策略。

端到端控制有限。如图 4 所示，我们几乎无法对运行时通信和服务行为进行端到端控制。A2A 等现有抽象提供了底层消息操作，却缺少用于在整个智能体流水线上进行高级协调或优化的钩子。例如，无法表达能够对交互式请求或延迟敏感型请求进行流水线级优先级排序的调度策略。也无法注入推测调用或提示 [1]（例如，预先迁移 KV 缓存以尽量降低运行时开销），从而引导工作流执行。

本质上，如今的协议和智能体服务系统简化了简单流水线的构建，却妨碍了适应与优化，尤其是在动态、高吞吐量和低延迟的服务环境中。

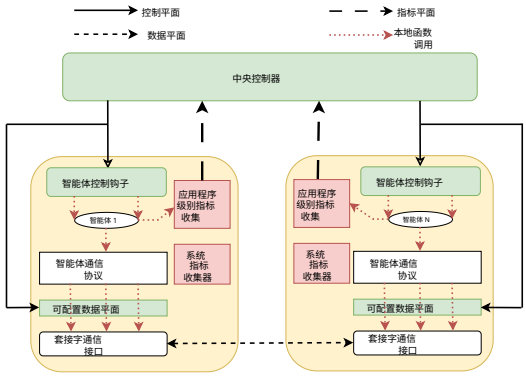


图5：我们的方案：控制平面根据全局遥测数据，同时编排数据平面和智能体/工具操作。

2.3 改进智能体服务

我们认为，要提升智能体流水线的服务能力，就需要从根本上重新思考智能体服务架构：根据当前智能体和系统状态，对智能体通信进行动态控制，以优化运营目标。要实现这一点，就需要引入新的智能体通信管理构建模块，以及面向工具、大语言模型智能体和服务系统的新接口，从而实现与这些构建模块的系统化集成。

3 软件定义方法

读者可能会熟悉上述需求——这让人联想到软件定义网络（SDN）如何为原本如黑盒般的路由器/交换机开放控制接口，从而实现更紧密的全网优化与管理。我们寻求对智能体服务进行类似的变革。具体而言，我们提议构建一个软件定义的智能体服务栈（图5）。

我们为高效的智能体推理提出了三个组成部分。第一，集中式控制平面，它根据数据做出决策，以控制每个智能体的通信及其他服务特性。第二，指标平面，用于收集数据，并将数据提供给控制器，以便其做出明智的决策。最后，数据平面基底使控制器能够进行细粒度控制。

3.1 智能体服务控制平面

受软件定义网络（SDN）的启发，控制平面负责解读高层运维人员目标，并在智能体服务中落实运行时决策。

目标。 我们设计控制平面以满足三个关键要求：(1)系统级可见性。控制平面必须能够访问指标平面（§3.2）公开的所有遥测数据——包括系统级负载、应用级性能和请求元数据。这为细粒度控制和端到端优化提供了空间。(2)

表 1：控制器所需的 API 接口

API	参数	描述
设置0	参数名称、值	将参数设置为指定值
reset()	参数名称	将参数设置为默认值

丰富的控制面。除了管理数据平面（§3.2）之外，以逻辑中央控制器形式实现的控制平面还应能够接入智能体或工具实现本身。这为运行时调整提供了更广阔的空间，从而提升整体性能和效率，例如重新调度、优先级分配、推测执行或跨节点状态传输。(3) 意图驱动控制。 负责管理智能体服务平台的基础设施工程师应能够指定高层目标（例如“在延迟约束下最大化吞吐量”），而控制平面则应将这些目标编译为具体的策略规则，使系统状态不断向目标演进。

为说明为何需要具备深入智能体级控制能力的丰富控制界面，请考虑这样一种场景：某个智能体实例变得过载。控制器可能会选择将请求重新路由到另一个实例。然而，如果该请求依赖于原始智能体持有的键值（KV）缓存状态，简单地重新路由就需要耗费大量资源进行重新计算，并增加延迟。通过接入智能体的运行时钩子，控制器可以在交接过程中传输相关状态，从而节省计算资源并提升响应速度。

丰富的控制界面很有价值，但也带来挑战——它需要与众多智能体和工具集成，而每个智能体和工具都有各自独特的 API。这通常需要定制控制平面 API，使控制程序难以构建，并要求频繁修改控制器。理想情况下，控制平面应通过统一而有效的接口简化集成。

设计。我们提出了一种控制平面设计，具有以下显著特点：

（*1*）数据平面控制。每个节点都运行一个可配置的垫片（如 §3.3 所述），用于接收并执行来自控制器的控制规则。控制器不会逐条管理消息，而是安装规则。规则分为两类：（a）智能体级规则规定智能体流水线的默认通信模式（例如，采用批处理还是流处理，以及在负载过高时仅接纳高优先级请求）。（b）请求级规则对请求路由到智能体实例的过程进行细粒度控制（例如，动态切换处理请求的智能体实例；或者，当智能体发送推测性请求时，在资源空闲前阻止该请求）。

(2) 智能体/工具钩子。每个智能体/工具在启动时都会公开一个注册接口，声明其支持的控制 API 和参数（例如暂停、限流、状态传输、调整优先级）。这些 API 使控制器能够执行有针对性的运行时操作，超越通信底层所能提供的能力。

然而，正如前文所述，与每个工具/智能体各自独特的 API 集成可能十分繁琐。为克服这一挑战，我们主张采用表 1 所述的标准化 API，所有工具和智能体都必须支持该 API。我们所提方案的 API 仅包含两个函数：“set”和“reset”。在注册时，每个智能体都会公开其拥有的各种控制 API 和参数旋钮。要使用这些旋钮，控制器会直接调用“set”函数，并传入参数名称和值。智能体或工具收到“set”调用后，必须更新该值，并自动确定要调用的函数。

例如，在智能体流水线中，为某个智能体设置最大批大小为 4；在 vLLM 中，对应的参数是 ‘max_num_seqs’。此时，我们的控制器会对相关智能体调用 set 函数："set(‘max_num_seqs’, 4)"。智能体负责实现垫片层，将 "set" 语句中的参数转换为相应的内部 API 调用，并更新批大小。这种关注点分离使控制器能够轻松扩展和演进，而智能体工作流开发人员只需实现一次特定的垫片层。

(3) 将意图驱动控制作为状态管理。控制器轮询数据（§3.2）时，会将数据填入逻辑状态存储。由运维人员编写并在控制器之上执行的控制逻辑，可从存储中获取相关状态视图。与 SDN 中的意图驱动网络管理类似 [7]，我们预计控制逻辑会实现算法来控制数据平面通信，使观测到的状态最终转变为目标状态，从而符合运维人员的预期目标。例如，“确保 90% 的交互式请求的端到端延迟在指定的服务级目标范围内”这一目标，可以转化为控制器轮询此类请求的延迟，并由控制逻辑进一步决定：(a) 将后台智能体降级为同步模式，(b) 重新分配 GPU 时隙，以及 (c) 转移关键请求的会话上下文，直至满足这些请求的服务级目标。

3.2 指标平面

有效的控制需要可见性。为了引导运行时的通信行为，控制平面必须同时观察系统级和应用级动态。不同于传统网络（交换机本地指标，如队列深度、流量速率，通常已足够），智能体服务需要更丰富的遥测数据，其中包括内存和计算资源利用率等系统级数据，以及查询完成时间和智能体依赖关系（条件式智能体执行）等应用级指标。此外，这些指标因智能体流水线的组件而异，例如，对于基于大语言模型的智能体服务，我们需要图形处理器利用率、每令牌耗时（TPT）和首令牌时间（TTFT）等指标。同时，

对于文件系统等工具，我们需要获取文件访问延迟、吞吐量和磁盘空间利用率。为控制复杂性，对不同指标的收集需要一个灵活而统一的指标平面。

目标。 指标平面应提供：(1) 在每个服务节点上，以低延迟、低开销的方式访问GPU/中央处理器利用率、内存压力和队列长度等关键信号。(2) 应用感知遥测，包括每请求延迟。(3) 灵活的聚合。不同的指标、智能体和工具将生成不同数量的数据；例如，下一个令牌生成时间在每次生成令牌时都会产生，而首令牌时间则在每次大语言模型查询中测量一次。指标收集平面需要根据控制器的要求提供各种聚合函数。(4)对指标的语义理解，使控制器能够恰当地解释这些指标并采取相应行动。例如，在优化吞吐量时，控制器需要认识到较高的 GPU 利用率是理想的，而大量的上下文切换则并非如此。

设计。 我们提出一种由以下部分组成的两层架构：(1) 每个节点上的本地指标收集器同时收集系统级（硬件、内核、运行时）和应用程序级（智能体工具使用情况、响应时间）遥测数据。这些数据存储在轻量级的共享内存结构中。(2) 通过集中轮询，控制平面按需获取指标。这样可以避免持续进行指标流式传输，并支持在需要时跨节点协调收集。(3)灵活的聚合函数。 为实现对各种不同指标的聚合，指标平面将提供编写自定义聚合函数的能力。

这种设计使控制平面能够从全局角度分析服务系统——识别计算、内存和通信热点；预测服务级别目标违规；并发现推测执行的机会[1]。通过将指标收集与决策分离，我们保留了模块化：智能体开发者可以公开领域专用指标，而无需重写控制逻辑。

为了实现对指标的语义理解，我们的解决方案是设计一种有限的规范语言，突出指标的行为。例如，创建收集指标函数的智能体工作流开发者可以编写一个与该指标关联的结构化文件（JSON 或 YAML）；控制器可以读取此文件，并利用其中提供的信息理解该指标的意义。然而，面对不同工具和智能体中的海量指标，要详尽地规范所有可能的指标颇具挑战。一种解决方案是让用户在收集指标的函数签名中用自然语言描述指标的意义；框架随后可以使用大语言模型将该描述转换为指标规范 [18]。

[1]控制器可以检测特定智能体的低负载，并利用它执行部分预填充或推测执行，从而缩短运行时间。

3.3 可配置数据平面

我们设想一种智能体通信数据平面，它将通信行为与静态应用逻辑解耦，并支持运行时自适应。

目标。 数据平面基底必须支持不同的消息粒度（例如，令牌级流式传输、消息级批处理）、优先级和速率控制策略。至关重要的是，数据平面必须提供一个简单的接口，以便系统级控制器能够从外部控制这些参数（§3.1）。例如，在智能体负载较高时，数据平面可以采用消息级发送，以摊销推理成本；而在交互式条件下，则可以切换回令牌流处理，以降低延迟。

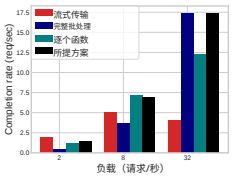
设计。我们不主张从头重新发明智能体通信协议，而是认为应将兼容性融入设计之中。我们所提方案的数据平面充当高层智能体协议（例如 A2A、MCP、ACP）与底层套接字或传输层之间的垫片层。这样既保留了开发人员使用熟悉且快速成熟的智能体应用程序接口的能力，又提供了一个新的控制点：可重新配置的垫片，用于调节消息属性。无论上层协议或底层传输的具体情况如何，数据平面都提供标准化的控制接口。

3.4 初步实现

我们实现了所提方案的原型，并基于 Google A2A 对一个工作流进行了实验。在该原型中，我们修改了 A2A 协议，使其调用我们定制的通信库；该通信库基于 gRPC 构建，支持同步、异步和流式通信。我们的指标平面会跟踪每个智能体/工具实例的当前负载。与之前一样，我们使用一个开发者智能体和一个测试智能体来运行 MetaGPT 工作负载。

通信控制。控制器旨在根据负载动态调整通信模式。如图 6 所示，我们的框架支持根据系统负载切换通信策略，从而快速收敛到最有效的机制。

扩展控制面。 为了展示让控制器访问底层智能体服务控制所带来的影响，我们修改了底层服务框架（vLLM [8]），提供了一种机制，可将 KV 缓存从一个智能体实例转移到另一个实例。我们启动了一个软件工程智能体实例和两个测试智能体实例，并对控制器进行编程，使其尽量减少测试智能体实例之间的负载不平衡。我们尝试了两种情况：一种是集成方案，控制器会预先向测试智能体发送提示，告知其某个请求



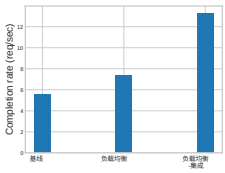


图 6：请求速率　图 7：带控制提示的控制器　集成：我们使用提示来控制消息通信粒度。在负载均衡时移动 K、V 缓存。

即将到达；另一种是请求到达之后，测试代理可以从另一个实例转移 KV 缓存。如图 7 所示，使用提示的效果 1.8×优于不使用提示。我们还观察到，所提方案的效果 2.3× 优于不进行负载均衡的基线。

4 相关工作

智能体式大语言模型系统因其迅速普及而受到广泛关注。大多数研究主要聚焦于三个领域：用于提示并提升智能体系统性能的算法；提高智能体服务效率；以及设计智能体与工具通信的协议。

算法改进。智能体式大语言模型算法仍在快速发展。这种算法演进可能会显著改变智能体应用程序流水线中的计算和控制流。例如，数据平面必须为这些调节参数提供简单的接口。MetaGPT [5],会为不同的大语言模型分配“开发者”“测试人员”“项目经理”等角色以生成代码；与此同时，近期提升代码生成能力的工作 [16] 要求能够访问实时环境。这种算法演进会显著影响工作负载，因为现在不同的智能体必须等待执行环境就绪后才能继续；具体等待时间取决于代码的复杂程度，可能长达数分钟。我们的框架控制平面可通过有效的资源编排，隐藏此类算法演进所带来的复杂性（例如，让多个并发请求共享图形处理器）。

系统改进。先前的研究 [9, 10] 致力于提升智能体系统的效率。然而，这些系统通常针对特定类型的智能体工作负载，并采用专门开发的策略来实现高效服务。我们的工作植根于软件定义网络（SDN）理念，可广泛应用于各种智能体工作流。

智能体通信。开发智能体通信协议一直备受关注——ACP [6],A2A [4],和 MCP [3] 是其中值得注意的例子——旨在实现不同智能体之间通信的标准化。在我们的工作中，我们并非提出一种新协议，而是展示智能体服务需要重新思考，并需要通过清晰统一的接口，围绕各种系统级目标（包括效率）实现有效的控制与管理。

5 讨论与结论

我们首先简要讨论本方案的自然扩展，然后探讨由此引出的有趣开放问题。

支持外部智能体。智能体工作流通常横跨用户管理的基础设施和由外部云托管的工具 []。我们的框架可自然扩展至此类混合部署。调用外部服务时，控制器可以将延迟、成本或速率限制纳入考量，选择兼顾性能与预算的配置。即使只是通过标准化 API 和轻量级适配层实现部分集成，也能带来可衡量的收益。

在线策略自适应。借助对请求上下文和系统指标的全局可见性，控制器可以在运行时调整策略，以减少服务级别目标违规。例如，如果流水线中的验证器触发了重新生成，控制器可以提高该请求的优先级，或将其重新路由以加快执行速度。这种实时调整能力是管理动态多智能体流水线的关键。

我们的方案是对智能体服务系统实施系统化控制的起点。除了围绕我们提出的接口和 API 展开的讨论外，我们的工作还提出了语言设计、控制器—智能体接口和指标基础设施方面的新问题。

智能体控制语言。我们设想采用声明式策略语言，让基础设施工程师能够表达高层目标，例如“延迟低于 200 毫秒时进行流处理”“负载高时避免推测调用”“队列深度较高时转移缓存”。与软件定义网络（SDN）不同，智能体系统所需的语义涵盖任务结构、延迟、智能体角色和计算资源。关键挑战包括定义足够丰富的原语、纳入实时跨智能体反馈，以及推理策略正确性。

控制器—智能体接口。有效的运行时控制需要控制器与智能体/工具之间更丰富的接口。要提供比我们当前方案更强控制能力的通用接口，需要具备更多功能，例如中断规划循环、克隆上下文或抑制非关键子调用。

智能体在结构和自主性方面各不相同，因此控制接口必须兼具灵活性和可预测性。一种可行的方案是定义能力类别（例如，无状态工具与有状态规划器），并为其制定标准控制 API。不过，能力发现、策略范围和冲突解决方面仍有一些悬而未决的问题。尽管如此，我们仍认为，采用标准 API 并让用户编写一个小型垫片层，以集成控制器 API

与智能体或工具的内部 API，将显著降低引入新智能体所需的工作量。

指标基础设施。我们的设计假设能够及时获取结构化遥测数据，但要大规模收集这类数据并非易事。首先，智能体服务涉及异构后端（vLLM、Triton、自定义运行时），因此难以实现一致的监测仪表化。其次，指标通常需要在长期运行的多智能体流水线中进行因果跟踪。第三，控制回路需要低延迟地访问海量遥测数据。尽管我们的初始原型提倡使用聚合函数来管理数据量，但我们仍需要新的抽象，以便向控制平面提供相关信号，并保持适当的保真度，同时避免系统不堪重负或削弱控制能力。

参考文献

[1] Saurabh Agarwal、Anyong Mao、Aditya Akella 和 Shivaram Venkataraman。Symphony：改进大语言模型推理工作负载的内存管理。*arXiv* 预印本 *arXiv:2412.16434*，2024年。[2] agntcy。一个促进智能体间协作的开源集体。https: //agntcy.org/，2025年。访问日期：2025年6月22日。[3] Anthropic。模型上下文协议。https://modelcontextprotocol.io，2025年。访问日期：2025年6月22日。[4] Google。一种开放协议，支持不透明智能体应用程序之间的通信和互操作。https://a2aproject.github.io/ A2A/latest/，2025年。访问日期：2025年6月22日。[5] Sirui Hong、Xiawu Zheng、Jonathan Chen、Yuheng Cheng、Jinlin Wang、Ceyao Zhang、Zili Wang、Steven Ka Shing Yau、Zijuan Lin、Liyang Zhou 等。MetaGPT：面向多智能体协作框架的元编程。*arXiv* 预印本 *arXiv:2308.00352*，3(4):6，2023年。[6] IDM。智能体通信协议。https://agentcommunicationprotocol.dev/，2025 22，2025年。访问日期：6月。[7] Xin Jin、Jennifer Gossels、Jennifer Rexford 和 David Walker。{CoVisor}：面向{软件定义}网络的组合式虚拟机监控器。载于第十二届 *USENIX* 网络系统设计与实现研讨会（*NSDI 15*），第87–101页，2015年。[8] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph Gonzalez、Hao Zhang 和 Ion Stoica。借助 PagedAttention 实现高效的大语言模型服务内存管理。载于第二十九届操作系统原理研讨会论文集，第611–626页，2023年。[9] Michael Luo、Xiaoxiang Shi、Colin Cai、Tianjun Zhang、Justin Wong、Yichuan Wang、Chi Wang、Yanping Huang、Zhifeng Chen、Joseph E Gonzalez 等。Autellix：面向通用程序形式大语言模型智能体的高效服务引擎。*arXiv* 预印本 *arXiv:2502.13965*，2025年。[10] Kai Mei、Xi Zhu、Wujiang Xu、Wenyue Hua、Mingyu Jin、Zelong Li、Shuyuan Xu、Ruosong Ye、Yingqiang Ge 和 Yongfeng Zhang。AIOS：大语言模型智能体操作系统。*arXiv* 预印本 *arXiv:2403.16971*，2024年。[11] Joon Sung Park、Joseph O'Brien、Carrie Jun Cai、Meredith Ringel Morris、Percy Liang 和 Michael S Bernstein。生成式智能体：人类行为的交互式模拟。载于第*36*届 *ACM* 用户界面软件与技术年度研讨会论文集，第1–22页，2023年。[12] Yujia Qin、Shihao Liang、Yining Ye、Kunlun Zhu、Lan Yan、Yaxi Lu、Yankai Lin、Xin Cong、Xiangru Tang、Bill Qian 等。ToolLLM：帮助大语言模型掌握{现实世界 API}。*arXiv* 预印本 *arXiv:2307.16789*，2023年。[13] Timo Schick、Jane Dwivedi-Yu、Roberto Dessì、Roberta Raileanu、Maria Lomeli、Eric Hambro、Luke Zettlemoyer、Nicola Cancedda 和 Thomas Scialom。Toolformer：语言模型能够自学使用工具。神经信息处理系统进展，36:68539–68551，2023年。[14] Yashar Talebirad 和 Amirhossein Nadiri。多智能体协作：发挥智能大语言模型智能体的力量。*arXiv* 预印本 *arXiv:2306.03314*，2023年。[15] John Chong Min Tan、Prince Saroj、Bharat Runwal、Hardik Mahesh-wari、Brian Lim Yi Sheng、Richard Cottrill、Alankrit Chona、Ambuj Kumar 和 Mehul Motani。Taskgen：一种基于任务、采用 strictjson 的记忆增强型智能体框架。*arXiv* 预印本 *arXiv:2407 15734*，2024。。[16] Xingyao Wang、Yangyi Chen、Lifan Yuan、Yizhe Zhang、Yunzhu Li、Hao Peng 和 Heng Ji。可执行代码操作能够激发更出色的大语言模型智能体。载于第四十一届国际机器学习会议，2024年。[17] Rui Yang、Lin Song、Yanwei Li、Sijie Zhao、Yixiao Ge、Xiu Li 和 Ying Shan。Gpt4tools：通过自我指令教会大语言模型使用工具。神经信息处理系统进展，36:71995–72007，2023年。[18] Xiaohu Zhu、Qian Li、Lizhen Cui 和 Yongkang Liu。大语言模型增强的文本到SQL生成：综述。*arXiv* 预印本 *arXiv:2410.06011*，2024年。