超越微服务：在 GPU 驱动的大语言模型工作负载上测试 Web 规模根因分析方法

Dominik Scheinert
logsight.ai GmbH 柏林，德国
dominik.scheinert@logsight.ai

Alexander Acker
logsight.ai GmbH 柏林，德国
alexander.acker
@logsight.ai

Thorsten Wittkopp 柏林工业大学 柏林，德国
t.wittkopp@tu-berlin.de

Soeren Becker l
ogsight.ai GmbH 柏林，德国
soeren.becker@logsight.ai

Hamza Yous 技术创新研究所 阿布扎比，阿联酋
hamza.yous@tii.ae

Karnakar Reddy 技术创新研究所 阿布扎比，阿联酋
karnakar.reddy@tii.ae

Ibrahim Farhat 技术创新研究所 阿布扎比，阿联酋
brahim.farhat@tii.ae

Hakim Hacid
技术创新研究所
阿布扎比，阿联酋
hakim.hacid@tii.ae

Odej Kao
柏林工业大学
柏林，德国
odej.kao@tu-berlin.de

arXiv:2603.02057v1 [cs.DC] 2 Mar 2026

摘要

大语言模型（LLM）服务已成为搜索、辅助和决策应用不可或缺的一部分。然而，与传统 Web 服务或微服务不同，支持大语言模型推理部署的软硬件堆栈复杂度更高，且经受的实地测试少得多，因此更容易发生难以解决的故障。要控制服务中断成本和服务质量下降，关键在于缩短平均修复时间；而在实践中，这取决于故障能否被快速识别、定位和诊断。自动化根因分析（RCA）通过识别发生故障的系统组件并追踪故障传播过程，加快故障定位。针对传统服务，人们已开发出大量根因分析方法，采用请求路径追踪、资源指标和日志数据分析。然而，现有根因分析方法并非为具有独特运行时特征的大语言模型部署而设计。本研究在受控故障注入条件下，评估了根因分析方法在遵循最佳实践的大语言模型推理部署中的有效性。在 24 种方法中——包括 20 种基于指标的方法、两种基于追踪的方法和两种多源方法——我们发现，多源方法的准确率最高，基于指标的方法的表现取决于故障类型，而基于追踪的方法大多失效。这些结果表明，现有根因分析工具无法泛化至大语言模型系统，因此有必要开发专门的分析技术并增强可观测性；为此，我们提出了相关指导原则。

CCS 概念

• **计算机系统组织** → **可靠且容错的系统和网络**；**可靠性**；可用性；云计算；　•**软件及其工程** → **软件性能**；软件可靠性；• **计算方法** → 自然语言处理。

关键词

分布式系统，可靠性工程，根因分析，智能运维，大语言模型

1 引言

大语言模型（LLM）作为搜索、辅助和决策应用的核心组成部分，其应用加速普及，这要求模型 API 始终可用，并能随着使用量增长而扩展 [10, 49]。随着大语言模型日益应用于电网、医学和国防等关键领域，系统停机可能造成严重后果，因此必须尽可能缩短 [3, 34]。向大语言模型的这一转变，从根本上改变了对系统基础设施的要求 [10]。大语言模型 API 需求不断增长，迫使基础设施团队迅速适应，在紧跟模型架构快速发展的同时，实时设计和扩展系统。尽管大语言模型推理基础设施的设计已开始趋向业内通用模式 [22, 40], ，但这些系统中的可观测性仍是一个亟待深入探索的领域。然而，可观测性是维护系统可靠性、可用性和安全性的基础。控制故障影响并维持服务质量，依赖于缩短平均修复时间（MTTR）；而这又取决于能否快速识别、定位和诊断故障。根因分析（RCA）在这一过程中发挥着关键作用 [9] ，能够大幅加快故障定位和解决。

现代微服务架构在规模、敏捷性和复杂性的需求驱动下，已取得显著发展。在过去十年中，涌现出许多自动化根因分析方法——主要面向云环境或微服务，包括因果图 [28, 52], 统计关联分析方法 [15], 和分布式追踪 [41] ——用于诊断和解决这些系统中的问题。基于大语言模型的服务通常集成在微服务架构中，为其他服务或用户提供用于查询模型的 API。然而，尽管其 API 遵循最佳实践，运行推理工作负载仍需要专用加速器硬件，并带来独特的运行特性，例如毫秒级动态请求批处理，以及用于张量传输的共享对象存储 [23, 33]。大语言模型请求处理本质上具有状态性，涉及临时键值缓存和调度器队列；在模型并行的情况下，单个请求还会经过多个设备。这些

特性改变了可观测性要求。传统以请求为中心的追踪数据以及统一的 CPU 或内存指标，正被碎片化的执行图、不透明的加速器运行时，以及嵌入 GPU 驱动程序中的硬件计数器 [22, 29]所取代。

目前尚不清楚，最初为由无状态Web服务组成的微服务架构设计的根因分析技术，在此类条件下是否仍然有效 [15, 58]。为此，本文系统评估了应用于大语言模型推理工作负载的先进根因分析方法。我们将具有代表性的因果关系图工具、统计工具和基于跟踪的工具部署在符合最佳实践的大语言模型推理技术栈上，并在受控故障注入条件下评估其性能[1]。尽管部分方法取得了良好效果，但我们的实验表明，大语言模型推理技术栈的特性——即新型加速器遥测数据、缺乏一请求一跟踪映射，以及复杂的系统组件相互依赖关系——给所评估的根因分析方法带来了泛化挑战。因此，最后我们概述了在加速器驱动系统中实现有效根因分析所必需的遥测增强和建模调整。我们的贡献如下：

- 使用受控故障注入，对分布式且具有代表性的基于参与者的服务进行测试，系统评估根因分析工具在大语言模型推理基础设施上的表现。
- 分析根因分析假设与现代、快速演进的 LLM 服务系统不匹配所引发的故障模式和误诊。
- 为图形处理器加速的大语言模型推理部署提供支持根因分析的可观测性设计指南。

概要。本文其余部分安排如下：第 2 节回顾大语言模型推理服务与传统微服务在架构考量和可观测性要求方面的相似之处与差异。第 3 节提出在大语言模型推理环境中有效开展根因分析的方法，并据此明确本文的研究范围。第 4 节介绍用于大语言模型部署的参考技术栈、可观测性配置，以及用于受控实验的细粒度异常注入流程。第 5 节呈现并讨论研究结果，最后制定指导方针。第6节介绍相关工作，第7节总结本文。

2 微服务架构中的大语言模型推理

要通过根因分析提升大语言模型推理的可靠性，了解大语言模型推理技术栈的架构特点，以及支持站点可靠性工程（SRE）运维所需的遥测数据，非常重要。

2.1 大语言模型推理服务的分层架构

随着大语言模型或基于大语言模型的服务在传统服务架构中的部署和使用日益增多， [24, 32, 44],由众多组件构成的异构环境变得更加复杂。如图 1 所示，大语言模型推理服务与传统微服务在架构上存在相似之处

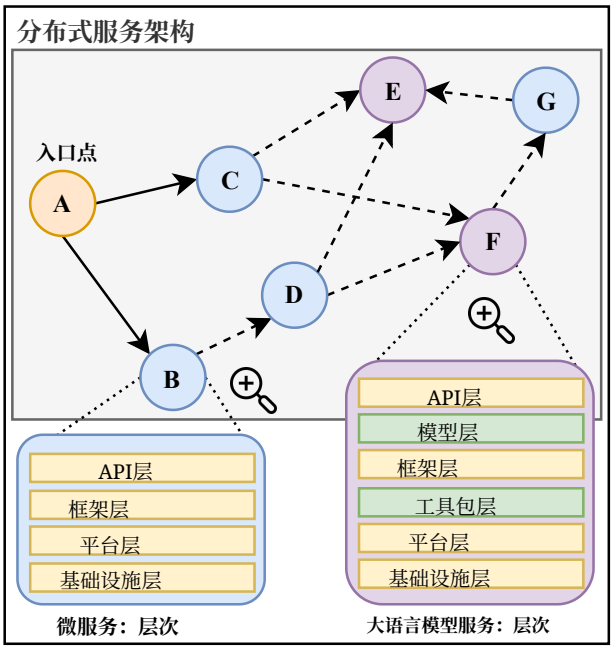

图 1：传统微服务和现代大语言模型推理服务作为分布式服务架构一部分的示意图。我们重点展示了它们各自分层架构的相似之处和差异。

但也表现出独特的特征。我们采用六层层次结构的概念， [58] 并探讨传统微服务与现代大语言模型推理服务之间的异同。

技术栈的底层是基础设施层，它提供计算所需的物理和虚拟资源，包括中央处理器、内存、存储、网络以及图形处理器和张量处理器等专用加速器。传统微服务和大语言模型服务都依赖这一层，但微服务工作负载通常受中央处理器限制，很少需要专用加速器。相比之下，大语言模型服务严重依赖图形处理器和张量处理器等加速器，以实现大规模提供大型模型服务所需的高吞吐量、低延迟推理性能。平台层负责为工作负载执行编排这些资源。在微服务和大语言模型架构中，该层都负责调度、生命周期管理和服务抽象，通常通过 Kubernetes 等编排系统实现。不过，大语言模型服务还会在这一层引入分布式机器学习计算框架（例如 Ray），这类框架专为跨多个图形处理器或节点扩展模型执行而设计。微服务通常作为独立单元运行，节点间协调极少；而当单个节点的资源不足时，大语言模型工作负载则需要借助这类框架来拆分并协调模型执行。

工具包层提供运行时库和算子集成，使应用程序能够利用专用硬件。对于大语言模型服务而言，这一层尤为重要，因为它们需要通过 CUDA 和 cuDNN 等库直接访问图形处理器的计算能力，还需要使用硬件算子（例如来自 NVIDIA、AMD 的算子），这些算子

[1]仓库链接：https://github.com/exalsius/rca-llm

负责在平台内安装和管理这些依赖关系。相比之下，传统微服务无需这些工具包，因为它们运行在通用 CPU 上，不与特定厂商的加速器驱动程序交互。框架层为实现实际应用提供了结构化方法。传统微服务使用通用应用框架（例如 Spring Boot）实现业务逻辑，而大语言模型服务则依赖 AI 框架（例如 Hugging Face Transformers）。这些框架提供模型定义语言、自动微分和经过优化的推理内核。因此，尽管两种架构都包含框架层，其用途却有根本区别：微服务侧重于执行业务逻辑，而大语言模型技术栈则侧重于高效的张量计算和模型推理。

模型层封装了参数检查点、分词器词汇表和提示模板等构件，它们共同定义了大语言模型学习到的函数。传统微服务中没有与此直接对应的层，因为传统微服务不会将学习到的模型作为运维构件进行管理。相反，它们可部署的单元通常是代码二进制文件或容器映像，并采用静态配置，而非学习到的参数。最后，API层向外部使用者提供功能。在架构上，传统微服务和大语言模型推理服务在此处趋于一致，因为两者都提供面向用户的 API、处理请求准入、执行路由并实施访问控制。然而，大语言模型服务集成了专门的推理逻辑（例如通过 vLLM 实现），包括连续批处理、令牌流式传输、缓存感知路由和上下文窗口管理等功能；这些功能超出了现代服务架构中微服务 API 常见的、通常无状态的请求—响应模式。

尽管现实世界中的服务有时会模糊这些边界，例如，框架可能自带服务网关，或工具包可能影响部署位置决策，但这种分层视图有助于明确不同架构中的职责。在这两种范式中，基础设施提供资源，平台负责管理资源；但大语言模型推理引入了不同的工具包、机器学习框架和模型层，并通过 API 层将它们作为统一端点提供。

2.2 可观测性要求

可观测性的核心是三种互补的信号类型，它们能够帮助诊断系统健康状况，并在发生故障时支持恢复。对于大语言模型服务，我们重新审视其核心挑战和关键考量，以建立可观测性。

指标捕获系统行为的连续、机器可读测量值，例如 CPU 利用率、内存消耗、网络吞吐量和请求延迟。在传统微服务中，指标主要在服务或容器级别跟踪资源使用情况、请求速率和错误计数，从而以最低的存储成本实现趋势分析和基于阈值的告警。对于现代大语言模型推理服务，还会公开或适合分析的其他指标，包括以 GPU 为中心的指标（例如内存带宽使用量、内核执行时间）以及首令牌时间（TTFT）和每秒生成令牌数等专用指标。这些指标对于识别基于加速器的推理工作负载中的性能瓶颈至关重要，而这些瓶颈无法通过仅 CPU 遥测观察到。

日志记录离散事件、警告和错误消息，提供指标数值聚合之外的叙述性上下文。在微服务中，日志通常包括 HTTP 请求日志、数据库查询和堆栈跟踪，可通过请求 ID 或跟踪上下文进行关联，以重建故障场景。至于大语言模型推理服务，它们通常会生成语义不同的日志，包括模型加载事件、GPU 分配决策、分词错误和框架级警告（例如 CUDA 内核故障）。确保日志中充实相关元数据（如模型标识符、提示长度和 GPU 设备 ID），对于诊断机器学习服务工作负载所特有的故障至关重要。

分布式跟踪将这些事件和指标串联起来，形成请求执行的端到端表示，捕获穿过服务、网关和处理阶段的因果路径。在微服务中，追踪数据映射多跳服务依赖关系、数据库查询和服务间调用，有助于诊断请求路径上的延迟峰值和瓶颈。对于大语言模型推理服务，追踪数据通常局限于推理服务边界内，捕获请求排队、批处理决策、模型调用和图形处理器执行跨度等节点内执行流程。这是因为整个推理工作流要么在单个服务内运行，跨节点服务调用有限；要么其分布式执行被 NVLink 等近期技术隐藏，从而对传统跟踪方法构成挑战。因此，大语言模型追踪必须揭示细粒度的服务内活动，而非服务间依赖关系。

在两种架构中采集这些遥测信号，都需要采用低侵入性的检测埋点，以避免性能下降。信号发出后，指标、日志和追踪数据会汇聚到可观测性平台，以支持后续数据分析任务。例如，在大语言模型推理期间检测到首词元时间升高等异常时，工程师通常会先查看观测到的指标偏差，再将其与 GPU 资源利用率指标相关联，检查框架和驱动程序日志以查找潜在错误，并分析跟踪跨度，以确定排队延迟或批处理行为欠佳等影响因素。总体而言，尽管微服务与大语言模型推理服务的可观测性核心理念一致，但这些信号的性质、粒度和解读方式可能存在显著差异。微服务主要关注服务间延迟和可用性，而大语言模型工作负载则需要遥测数据来捕捉加速器利用率、框架级执行情况以及特定于模型的性能指标。结合图 1，要全面观测大语言模型推理服务，就应整合从基础设施和平台到工具包、框架、模型及 API 各层的信号，从而全面了解正在发生的故障并有针对性地加以缓解。

3 LLM 推理的 RCA 方法

生产级 LLM 推理系统在设计上就是分布式系统。要确保分布式系统的可信赖性，必须高效地检测、定位并缓解故障。可信赖性涵盖可用性、可靠性和可维护性等属性 [5], 其中，可用性通常定义为系统能够提供正确服务的时间占比：

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}.$$

这里，MTBF 表示平均故障间隔时间，MTTR 表示平均修复时间。因此，降低 MTTR 成为生产系统的一项运维目标 [7] 。

分布式系统中的故障很少只局限于单个组件。由于存在相互依赖关系，局部故障可能会传播，导致级联故障，进而降低整个系统的性能或可用性 [13]。快速识别此类故障的源头对于限制其影响至关重要。站点可靠性工程（SRE）通过运用可观测性数据来快速检测、定位和缓解故障的实践，将这一目标付诸实施 [9]。

在这一工作流程中，根因分析（RCA）发挥着重要作用。RCA 旨在识别其故障引发了所观察到故障（例如令牌生成缓慢）的组件或组件组。形式上，设一个分布式系统由 N组件 $\mathrm{C} = \{c_1、\ldots\ldots、c_N\}$ 构成。每个组件 c_i会随时间 M发出遥测信号 t，从而形成多变量时间序列$\mathbf{X}_t^i = \left(x_{(i,1)t}、\ldots\ldots、x_t^{(i,M)}\right)$.。给定发生在时间 t_F 的观测故障，RCA 会分析观测窗口$[t_0, t_{\text{rca}}]$ 内的遥测数据，其中 $t_0 < t_F < t_{\text{rca}}$，以推断导致故障的组件子集 $c^* \subseteq C$。在实践中， t_0 的选取旨在使遥测数据量保持在根因分析方法可处理的范围内，或者受时间序列数据库保留策略的隐式约束。

许多自动化根因分析方法会围绕指标、日志或追踪数据构建依赖关系图 $\mathrm{G} = (V, E)$，以建模系统结构和信息流 [37]。随后应用评分函数 $s\colon V \to \mathbb{R}$，对最有可能解释故障事件的节点（例如指标或组件）进行优先级排序。总体而言，根因分析面临高维稀疏标注数据、监督有限、因果关系之间存在时间延迟以及可观测性不完整等挑战。因此，有效的根因分析技术必须能够从嘈杂、不完整的遥测数据中推断因果结构或相关性。

3.1 研究目标

尽管传统微服务应用中的根因分析已得到广泛研究，但其在现代大语言模型推理服务中的适用性仍鲜有探讨。如前所述，这些服务在多个维度上可能存在差异。然而，现有的根因分析方法

主要是围绕传统微服务设计的。目前尚不清楚，现有根因分析方法的假设和方法论是否适用于大语言模型服务环境；在这些环境中，由于架构差异，故障可能以其他方式显现。这促使我们开展本研究，系统评估 24 种根因分析方法，测试其性能，以识别符合业界和学术界最佳实践的大语言模型推理技术栈中四种故障场景的正确根因。

4 为大语言模型推理架构启用根因分析

随着大语言模型服务持续成为一个新兴领域，且标准化解决方案寥寥无几，我们开发了一个实验系统[2]，该系统 [23, 33, 40] 实现了当前可扩展推理的最佳实践，同时

[2]仓库链接：https://github.com/exalsius/rca-llm

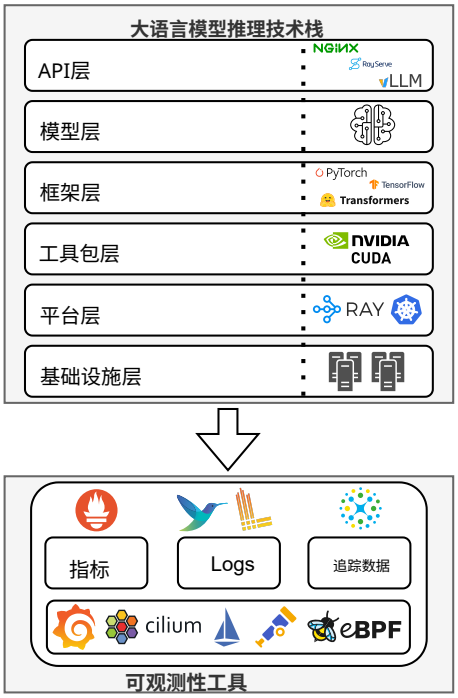

图 2：我们的大语言模型推理部署技术栈和可观测性工具的高级概览。第 2 节中此前讨论的不同架构层以及遥测数据支柱，被映射到具体技术，最终构成我们的实验系统，并为其他实践者和研究人员提供蓝图。

支持详细的可观测性和故障注入。该系统为在受控条件下评估根因分析方法提供了一个逼真的基础。下文将介绍其架构，并将其映射到前文讨论的大语言模型推理层，同时详细说明集成的可观测性工具以及用于实现系统化评估的自动化异常注入工作流程。在可行的情况下，我们使用 Ansible 例程和社区维护的 Helm 图表，以确保可复现性。图 2 展示了整体概览。

4.1 大语言模型部署参考技术栈

对于典型大语言模型推理架构的不同层，我们选择了以下技术：

基础设施。我们的基础设施由从一家公有云服务提供商租用的八台虚拟机组成，每台配备 32 个虚拟 CPU、一块 NVIDIA A100 PCIe 80GB GPU 和 240GB 内存。这是中小型自托管大语言模型集群的典型配置：近期同行评审的系统研究评估了在单块 A100 80GB 主机和小型集群上提供大语言模型服务的情况 [11, 42], ，并在类似生产环境的实验中使用了 A100 80GB 云实例 [1, 57]。这使我们能够在支持数据并行和模型并行的同时，试验动态资源分配。

平台。 所有机器均使用 Kubernetes [46], 进行编排，具体采用轻量级的 K3s[3] 发行版。Kubernetes 管理计算资源（Pod、部署）、存储资源（卷、密钥、ConfigMap）和网络资源（服务、入口）。此外，平台层通过将 Kubernetes 与 Ray [33], 集成来实现。Ray 是一个开源分布式计算框架，提供用于数据处理、训练和推理的可扩展 API。Ray 通过 kuberay-operator[4],与 Kubernetes 紧密集成；该运算符定义了 RayCluster 和 RayService 等自定义 Kubernetes 资源。RayCluster 定义 Ray Pod（头节点和工作节点）的组成及扩缩容方式，而 RayService 则将集群配置与部署逻辑结合起来。我们通过 NGINX 网关向外部公开 RayService 创建的服务。此配置支持在多节点、多 GPU 环境中进行分布式大语言模型推理。

工具包。 我们使用 NVIDIA GPU Operator[5] 自动配置 GPU 支持所需的 CUDA、驱动程序和运行时组件。该 Operator 可确保在 Kubernetes 中正确发现和配置图形处理器，从而简化这一通常容易出错的流程。

框架。 我们依赖基于 PyTorch 的工具，尤其是 Hugging Face Transformers，以便与预训练模型进行交互。尽管 Transformers 并未直接纳入我们的参考技术栈，但它是推理引擎 vLLM 用于处理模型的底层库。 [51]

模型。 模型管理 [58] 通过 Hugging Face 仓库实现，这些仓库提供对 TII Falcon、通义千问或 Llama 等提供商的预训练大语言模型的访问。

API。 API层由 vLLM 组件构建。vLLM 是一个用于基于 Transformer 的模型的快速高效服务库。vLLM [23]支持连续批处理和兼容 OpenAI 的 API 等高级功能。它与 Ray Serve 集成，以支持分布式服务和推理。NGINX 网关根据 URL 路径中的模型 ID，将传入请求路由到 Ray Serve Pod。请求会分发到可用副本，每个副本都运行一个 vLLM 引擎实例，负责预处理、推理和后处理。这种架构可提供可扩展的多节点大语言模型服务，并实现高吞吐量。

4.2 大语言模型推理部署的可观测性

我们通过添加涵盖指标、日志和追踪数据的可观测性解决方案，扩展了大语言模型推理技术栈，并将其集成到基于 Kubernetes 的部署中。我们还使用搭载基于 eBPF 的 Cilium[6] 容器网络接口（CNI）插件的 Kubernetes，以增强网络层面的可观测性。

指标。 我们使用 Prometheus[7] 及相关工具收集指标。此配置可采集系统级指标（例如 CPU、内存）以及来自 Ray 和 vLLM 的应用特定指标。Prometheus 会抓取由 Kubernetes PodMonitor 和 ServiceMonitor 资源定义的公开端点。我们还部署 Istio[8] 作为服务网格，采用其 ambient 部署模式（ztunnel + way-point 代理），并收集其遥测数据，以监控大语言模型部署组件之间的网络通信。这包括各服务的请求速率、延迟和错误指标。收集到的关键大语言模型推理信号（如 TTFT 和端到端延迟）使用 Grafana[9]进行可视化。指标还会关联 Kubernetes 元数据，以便进行精细筛选和调试。

日志。Ray 和 NGINX 网关的应用程序日志通过 Fluent Bit[10] 边车容器采集。边车容器从共享卷读取日志，并将其转发到 Loki[11] 进行持久化。此方法无需修改应用代码，并支持结构化日志聚合。日志会附加元数据标签（例如 Pod 名称、命名空间），从而能够与指标和追踪数据有效关联分析。

追踪数据。 为了在不插桩应用代码的情况下捕获分布式跟踪，我们使用 DeepFlow [41], 一种基于 eBPF 的追踪解决方案。DeepFlow 代理以 DaemonSet 的形式在 Kubernetes 集群中运行，并挂接到内核级事件。这使得我们能够以极低的开销 [50] 对端到端请求路径进行零侵入追踪。追踪数据会用元数据（例如 X-Request-ID 等 HTTP 标头）加以丰富，并通过自定义插件在 Grafana 中可视化。这使我们能够在分布式大语言模型服务流水线的各个组件中定位故障。

总体而言，我们的云原生可观测性技术栈支持在类生产环境的大语言模型部署中进行实验与故障排查。此外，它还能收集遥测数据，为我们系统化的根因分析方法评估提供支撑。一系列冒烟测试验证了这一点，其中一项测试的结果如图 3 所示：所构建的大语言模型推理技术栈能够处理非平凡负载，同时接受我们可观测性解决方案的全面分析。

4.3 负载生成与故障注入

为了系统地评估大语言模型推理基础设施中的根因分析工具，我们需要运行真实的负载场景，同时向已部署的推理服务注入可控故障。

目标系统。 在不失一般性的前提下，我们按照第 4.1 小节所述的推理技术栈，部署 Falcon-H1-7B-Instruct 大语言模型；其中，RayCluster 的每个 Pod 均可使用一个图形处理器、八个 CPU 核心和 16GB 内存。RayCluster 总共由一个 Ray 头节点 Pod 和三个 Ray 工作节点 Pod 组成。该模型采用四路数据并行部署，这意味着 RayCluster 的每个 Pod 都负责管理并运行一个模型副本。值得注意的是，具体选择哪种大语言模型与本研究的范围无关，因为我们研究的是整个大语言模型推理技术栈，并将所服务的模型视为可互换的工作负载。

负载生成。 为了评估系统在负载下的表现，我们使用基于 Locust[12], 构建的自定义基准测试工具，以每秒 5 次查询（QPS）的固定查询速率运行，最多使用 300 个工作进程。每次测试持续 360 秒，并向已部署的大语言模型的推理端点发送一系列聊天请求。每个查询平均包含 512 个提示词令牌，并生成 256 个流式输出令牌，代表典型的问答聊天 [12],

[3] https://k3s.io/, 访问时间：2026年3月 https://github.com/ray-project/kuberay,访问时间：2026年3月 [5] https://github.com/NVIDIA/gpu-operator, 访问时间：2026年3月 [6] https://cilium.io/, 访问时间：2026年3月 [7] https://prometheus.io/, 访问时间：2026年3月 [8] https://istio.io/, 访问时间：2026年3月

[9] https://grafana.com/, 访问时间：3月 2026 [10] https://fluentbit.io/, 访问时间：2026 https://grafana.com/oss/loki/, 访问时间：3月 [11] [12] https://locust.io/, 访问时间：2026年3月

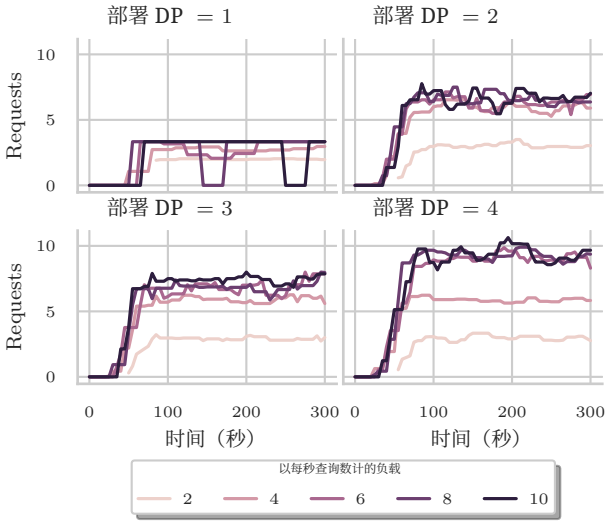

图 3：我们展示了已开展冒烟测试的示例结果，这些结果展示了在不同数据并行（DP）配置下，面对恒定的每秒查询数负载时，随时间处理的请求数量。

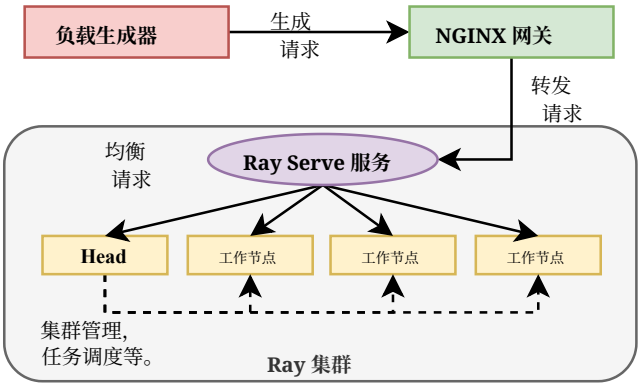

图 4：实验设置与请求处理工作流的抽象示意图。负载生成器向从集群对外暴露的 NGINX 网关发送请求。网关随后将请求转发至负责为特定大语言模型均衡服务请求的 Kubernetes 服务。最终，该服务将请求发送至某个可用端点，该端点包含单个 Ray 头节点 Pod 以及任意数量的已连接 Ray 工作节点 Pod。

与我们部署中使用的兼容 OpenAI 的 API 相匹配。为限制请求并发数，我们将同时运行的请求数量上限设为 500。如图 4 所示，此设置使我们能够在受控负载条件下，针对系统可扩展性、吞吐量和延迟建立性能基线，然后继续研究可观测性技术栈和已部署的根因分析方法如何响应合成生成的性能瓶颈。

异常注入。 我们实现了相关工作 [52] 中提出的三种 Pod 级故障注入实验，并使用 ChaosMesh[13] 将其集成到大语言模型推理技术栈中。此外，我们还实现了一种会引发 GPU 压力的 GPU 级故障，形成以下四种故障场景：

- **CPU 占用**：在目标 Pod 中启动 16 个线程，以满负载运行（StressChaos CRD）。
- **内存泄漏**：在目标 Pod 中随时间逐步额外分配 8GB 内存（StressChaos CRD）。
- **网络延迟**：将所有进入目标 Pod 的流量延迟一秒（NetworkChaos CRD）。
- **GPU 降频**：使用 nvidia-smi 工具将功率限制从 300W 调至 150W，以降低连接到目标 Pod 的图形处理器性能。

我们按顺序向 Ray 工作节点 Pod 注入故障，每次持续一分钟，并在其间设置冷却期以恢复正常行为。尽管真实部署中的故障可能同时出现（例如，GPU 降频与网络延迟同时发生），我们仍遵循既有的可信性基准测试实践，一次只注入一个故障。顺序单故障注入可确保实验可控、可复现，并能将观测到的异常明确归因于所注入的原因。先前的工作在故障注入与混沌工程同样强调，在考虑更复杂的多故障场景之前，应将单故障实验作为方法学基线 [6, 8, 18]。该设置使我们能够利用前文讨论的可观测性系统创建丰富的数据集。一旦采集到数据集，我们的目标是研究既有的根因分析方法如何推广到大语言模型系统独特的遥测数据与故障模式。

5 实验结果

在本节中，我们展示实验结果[14]，，分析现有根因分析方法在受控故障场景下应用于大语言模型推理工作负载时，依据所选评估指标所体现的性能与局限性。

5.1 评估指标

为量化所选基线方法在注入大语言模型推理系统的一组异常 A 上的性能，我们采用文献中的两项标准指标，即 $AC@k$ 和 $Avg@k$。前一项指标表示算法返回的前k个结果中包含实际根因的概率。较高的 $AC@k$分数，尤其是在 k取值较小时，表明该算法能在排序较靠前的位置正确识别根因。对于异常 $a \in A$，令 $R^a[i]$ 表示排名第 i的服务，v^a_{rc} 为真实根因集合。则 $AC@k$ 定义为：

$$AC@k = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \frac{\sum_{i<k} \left(R^a[i] \in v^a_{\text{rc}}\right)}{\min(k, |v^a_{\text{rc}}|)} \tag{1}$$

因此，$Avg@k$衡量根因分析方法的整体性能，其定义为 $Avg@k = \frac{1}{k}\sum_{j=1}^{k} AC@j$。解读这些评估指标时，请注意，根因分析方法需要在六个候选 Pod 中进行排序（三个 Ray 工作节点、一个 Ray 头节点、一个

[13]https://chaos-mesh.org/，访问时间：2026年3月

[14]代码仓库链接：https://github.com/exalsius/rca-llm

NGINX 网关和一个 KubeRay Operator），因此随机基线为 $AC@1 \simeq \frac{1}{6}$。

5.2 根因分析方法的性能分析

我们基于一个开源根因分析基准测试开展工作，该基准测试也已在近期发表的研究中提出并使用 [37, 38]。它包含多种先进的根因分析方法——其中大多数基于指标运行（例如 CIRCA [25], CausalAI [4]），但也有一些基于追踪数据（例如 TraceRCA [26], MicroRank [56]），甚至采用多源方法（例如多源 BARO [36], PDiagnose [25]）。我们对该基准测试框架进行了调整，使其也能使用并处理我们在实验中收集遥测数据后生成的数据集。此外，我们还集成了另外三种根因分析方法：MicroRCA [52], Microscope [28], 和 MonitorRank [21],，这些方法也常用于相关工作中的比较。这样一来，我们总共获得了 24 种基线方法。每种根因分析方法都会接收① 注入异常前五分钟时间窗口内的遥测数据、 ② 注入时间戳，以及 ③ 一分钟注入区间内的遥测数据。

我们在表 1 中展示了实验的主要结果。这里，我们总结了所有基线方法在不同故障类型下、按照前文介绍的评估指标得出的结果。每种故障类型都分别注入到三个 Ray 工作节点 Pod 中，每个 Pod 注入十次，因此每种故障类型共注入 30 次，所有故障类型合计注入 120 次。需要注意的是，除了随机选择策略和所有明确列出的根因分析方法外，我们还纳入了多种方法，它们实际上是因果发现方法（Peter-Clark（PC） [45]；快速因果推断（FCI） [45]；Granger [16]；LiNGAM [43]）与评分方法（PageRank（PR）；随机游走（RW））的组合。在极少数根因分析方法无法收敛的情况下，我们默认采用随机选择策略来处理该错误。

基于指标的方法。与所采用的根因分析基准测试及相关工作一致，基于指标的方法会获得每个 Pod 的通用指标，包括 CPU 利用率、内存消耗、网络流量和延迟。此外，为了考虑 AI 工作负载的特殊性，我们还纳入了 GPU 利用率信息。如果相应的根因分析方法支持，我们还会提供一个从网络通信中推导出的有向无环图（DAG）对象，以全面呈现 Pod 之间的依赖关系。

表 1 显示，基于指标的根因分析方法在不同故障类型上的表现参差不齐。PC-PR、FCI-PR 和 Granger-PR 等经典因果发现方法的 $Avg@5$ 得分中等（例如，FCI-PR：CPU 为 0.57，网络为 0.59），但其 $AC@1$ 仍然较低（≤ 0.33），这表明虽然真正的根因通常位于前 5 个候选对象之中（总共 6 个 Pod），但很少排在首位。这说明这些方法可能缺乏明确的排序能力，这一观察结果与此前对微服务系统的评估一致。相比之下，NSigma 等现代方法表现近乎完美，在 CPU 上达到 $AC@1 = 1.00$，在内存和网络上达到 ≥ 0.70。同样，BARO、CIRCA 和 RCD 的排序准确率也很高，例如，CIRCA 的 $AC@1$ 在内存上达到 0.80，在网络上达到 0.70，而 RCD 的 $AC@1$ 在 CPU 上达到 0.50。值得注意的是，这些方法在大多数故障类型上的 $AC@3$（≥ 0.80）和 $Avg@5$（≥ 0.69）也保持较高水平。它们能够将真正的根因排在最前或接近最前的位置，这表明在候选集较小时，其实际效用超出了单看 $Avg@5$ 所能体现的程度。然而，LiNGAM-PR/RW 和 MicroCause 等方法的表现持续不佳，常常将运维组件或网关等无故障 Pod 错误排在前面，$AC@1$ 接近于零，且 $Avg@5$ 低于 0.25，表明它们在大语言模型推理中的适用性有限；这与它们在微服务基准测试中的表现一致。

基于追踪的方法。与所采用的根因分析基准测试及相关工作一致，基于追踪的方法会获得有关各个请求的追踪信息，包括追踪 ID、跨度 ID、父跨度 ID、服务名称、方法名称、操作名称、状态码、持续时间以及其他详细的时间相关信息。

基于追踪的方法效果有限。TraceRCA 的 $Avg@5$ 接近基线水平（0.28–0.55），而 $AC@1$ 低于 0.05，这表明它很少能将真正的根因识别为排名第一的候选项。MicroRank 在所有故障类型上的表现都处于基线水平，$Avg@5 = 0.20$，且 Top-1 准确率为零，这证实了仅基于追踪的方法难以适用于大语言模型推理场景：与多层微服务相比，请求传播不仅极少，而且考虑到大语言模型推理系统的请求批处理策略，其处理方式也有本质区别。进一步检查后，我们发现，真正的根因经常被错误归因于 NGINX 网关，因为症状会在该组件上显现（例如，由于请求队列或延迟累积）。

多源方法。与所采用的根因分析基准测试及相关工作一致，多源方法除了使用指标和追踪数据（如前文所述）外，还会使用日志数据（由时间、服务名称和日志消息组成）。此外，还会根据追踪数据计算高层统计信息（各服务和操作在不同时间窗口内的错误数／成功数），并根据日志数据计算高层统计信息（各服务在不同时间窗口内的日志数）；如果方法支持，还会将这些统计信息提供给相应方法。

多源根因分析方法往往优于许多仅基于指标和仅基于追踪的方法。MM-BARO 的 $AC@1$ 最高可达 0。77（内存），网络上的 $Avg@5$ 为 0。87。PDiagnose 在各种故障类型上的 $AC@3$ 和 $Avg@5$ 也同样出色，但其 $AC@1$ 仍然较低（在所有故障类型上基本为零）。这些结果与以往研究一致，表明整合指标和追踪数据能够提升根因分析性能。不过，鉴于候选集较小，这些接近上限的 $Avg@5$ 分数不如 $AC@1$ 和 $AC@3$ 的显著提升那么有说服力；后两者表明排名最高的预测更加可靠。

5.3 错误归因模式的关键发现

为了深入理解表 1 中展示的汇总结果，我们分析了部分根因分析方法的错误归因模式（每个类别选取两种方法；对于基于指标的方法，我们选取了表现较有潜力的方法）。相应结果见图 5。在所有受评估的方法中，异常定位方面的系统性错误显而易见。具体而言，六种方法中的五种，即 BARO、NSgima、MicroRank、TraceRCA 和 MM-BARO，即使注入的故障源自 Ray 工作节点，也会经常将责任归咎于 NGINX 网关。这种行为表明，这些方法明显偏向网关级组件。剩下的方法 PDiagnose 则没有将错误归咎于网关

表 1：24 种不同基线方法在四种故障类型上的根因分析性能。

Data 来源	方法	CPU			内存			网络			GPU		
		AC@1	AC@3	Avg@5	AC@1	AC@3	Avg@5	AC@1	AC@3	Avg@5	AC@1	AC@3	Avg@5
指标	随机	0.17	0.50	0.51	0.10	0.43	0.47	0.30	0.57	0.58	0.23	0.63	0.57
	PC-PR	0.10	0.43	0.38	0.20	0.53	0.52	0.07	0.47	0.45	0.33	**0.67**	0.60
	PC-RW	0.33	0.57	0.66	0.33	0.47	0.63	0.33	0.60	0.66	0.33	0.43	0.61
	FCI-PR	0.23	0.53	0.57	0.17	0.40	0.44	0.33	0.53	0.59	0.07	0.53	0.49
	FCI-RW	0.33	0.57	0.66	0.33	0.47	0.63	0.33	0.60	0.66	0.33	0.43	0.61
	Granger-PR	0.23	0.37	0.45	0.13	0.57	0.53	0.20	0.57	0.57	0.23	0.57	0.55
	Granger-RW	0.33	0.57	0.66	0.33	0.47	0.63	0.33	0.60	0.66	**0.37**	0.53	**0.65**
	LiNGAM-PR	0.00	0.03	0.09	0.00	0.13	0.23	0.00	0.17	0.25	0.10	0.53	0.49
	LiNGAM-RW	0.33	0.57	0.66	0.33	0.47	0.63	0.33	0.60	0.66	0.33	0.43	0.61
	CausalRCA [53]	0.20	0.57	0.59	0.20	0.40	0.43	0.17	0.40	0.45	0.23	0.57	0.57
	CausalAI [4]	0.33	0.47	0.43	0.37	0.50	0.48	0.23	0.23	0.23	0.03	0.07	0.05
	MicroCause [31]	0.00	0.07	0.12	0.00	0.07	0.14	0.00	0.07	0.13	0.03	0.20	0.27
	ϵ诊断 [39]	0.27	0.70	0.57	0.20	0.43	0.37	0.40	0.70	0.63	0.17	0.53	0.45
	BARO [36]	0.13	**1.00**	0.83	0.77	0.93	**0.91**	0.33	**1.00**	0.87	0.20	**0.67**	0.59
	RCD [19]	0.50	0.80	0.73	0.43	0.77	0.69	0.43	0.83	0.73	0.23	0.53	0.46
	CIRCA [25]	0.40	**1.00**	0.86	**0.80**	0.93	**0.91**	0.70	**1.00**	0.93	0.20	0.63	0.61
	NSigma [28]	**1.00**	**1.00**	**1.00**	0.70	0.90	0.87	**0.93**	**1.00**	**0.99**	0.17	0.63	0.55
	MicroRCA [52]	0.20	0.50	0.51	0.13	0.37	0.46	0.60	0.90	0.81	0.07	0.50	0.42
	Microscope [28]	0.20	0.73	0.56	0.30	0.87	0.65	0.63	0.90	0.82	0.13	0.53	0.41
	MonitorRank [21]	0.13	0.27	0.33	0.03	0.10	0.21	0.27	0.77	0.69	0.07	0.43	0.35
追踪	MicroRank [56]	0.00	0.00	0.20	0.00	0.00	0.20	0.00	0.00	0.20	0.00	0.00	0.20
	TraceRCA [26]	0.03	0.67	0.55	0.00	0.00	0.01	0.00	0.30	0.28	0.00	0.20	0.15
多来源	MM-BARO [36]	0.13	**1.00**	0.83	0.77	0.93	0.89	0.33	**1.00**	0.87	0.10	0.37	0.41
	PDiagnose [17]	0.00	**1.00**	0.79	0.00	**1.00**	0.80	0.00	0.87	0.71	0.00	0.57	0.58

而是将故障归因于 Ray 头节点 Pod，这表明它也倾向于归咎于较高层级的组件，而非实际发生故障的 Ray 工作节点，因为 Ray 头节点 Pod 也参与传入请求的负载分配。值得注意的是，对于 TraceRCA，真正的根因甚至经常不在

由于该方法内部应用了筛选步骤，根因候选项被过滤，导致出现“未找到”的情况。

总体而言，在所有故障类型中，受检方法均未能可靠地将 Ray 工作节点 Pod 识别为异常来源（ *AC*@1 ）。虽然这些发现清晰地揭示了各方法在受控实验条件下的局限性，但也引发了有关实际部署的重要问题。尤其是，观察到的偏差表明，运维故障排查可能存在效率低下的问题，也凸显出有必要重新审视根因分析技术应如何适配分布式微服务环境，而这类环境中正日益包含基于大语言模型的推理服务。下文将讨论这些影响。

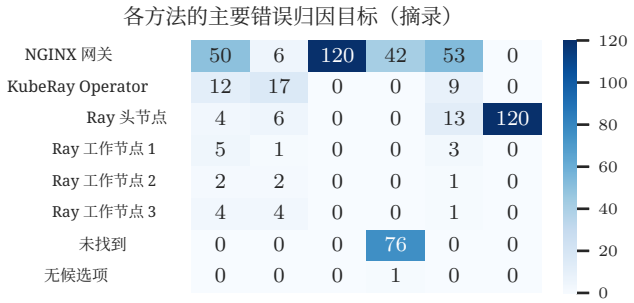

图 5：不同类别中所选根因分析方法的误归因次数。大多数方法会系统性地将故障归因于上游组件（NGINX 网关或 Ray 头节点 Pod），而非实际发生故障的 Ray 工作节点 Pod。

5.4 讨论

实验评估揭示了自动化根因分析在大语言模型推理工作负载场景中的潜力与局限性。具体而言，系统在不同故障模式下的表现（参见图 6）揭示了此类部署场景中的可观测性需求以及故障传播的特性。

CPU 异常（如合成 CPU 高负载进程）会对推理性能产生一致且可测量的影响。这些异常会降低多种对延迟敏感的操作的性能，包括请求调度、分词，以及 CPU 与图形处理器之间的数据传输。因此，它们会清晰地反映在响应延迟和令牌生成时间等指标中，因而相对容易定位。

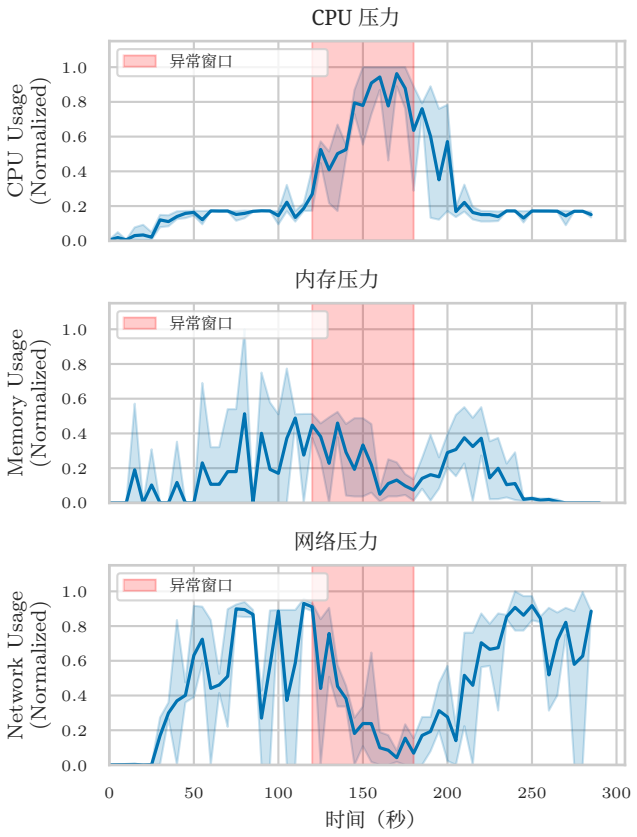

图 6：故障注入对系统指标的影响。CPU 占用过高会在注入期间自然导致 CPU 使用率随时间上升。针对延迟的网络异常会暂时降低整体通信量。对于内存，我们首先观察到使用量下降，这可能是内存溢出情况所致；随后，当 Ray actor 恢复时，内存使用量再次上升。GPU 降频（图中未显示）由于静态分配，并未导致 GPU 利用率发生显著变化。

此外，CPU 异常可以通过可复现且不会造成干扰的方式注入，这有助于将其用于测试流水线。相比之下，内存相关异常往往会因内存不足（OOM）导致 Ray actor 突然终止。在这些故障发生后，特定模型副本会暂时不可用，直到 actor 重启。这会导致部署出现部分性能下降，有时还会造成整体服务暂时不可用。重要的是，这类问题的诊断过程更为复杂，因为内存管理和 actor 监控依赖于框架特定机制，而这些机制并不总是透明的，也难以通过标准系统指标轻松观测到。这凸显了在可观测性技术栈中纳入运行时日志和大语言模型专用遥测数据的重要性。网络异常在故障注入和诊断方面都带来了最大的挑战。在实践中，我们发现，一些根因分析方法在此类故障场景下往往无法收敛，这可能是由于遥测数据不完整，因为 Pod 级网络混沌也会影响边车数据采集容器。实验过程中经常需要人工干预来重启并恢复，这限制了自动化程度。通过功率限制引发的 GPU 异常会减慢令牌生成速度并增加响应时间，反映出模型执行性能下降。然而，根因分析方法未能检测到这些异常，因为静态分配使利用率和内存等标准 GPU 指标保持不变，这暴露了当前遥测数据和根因分析方法的不足。最后，在一系列探索性附加实验中，我们在 API 级别注入 HTTP 延迟，导致 Ray actor 意外故障且恢复行为不稳定，因此决定不将这些结果纳入论文。尽管如此，这些初步结果仍凸显了分布式 LLM 系统在 API 级行为受到扰动时的脆弱性 [20],，并进一步增加了根因分析的难度。

展望未来，我们的研究结果凸显了对 LLM 专用故障诊断的需求。NSigma、CIRCA、BARO 和 RCD 等现代根因分析方法在大多数注入故障下表现良好，而经典因果方法和基于追踪的方法则常常会误判无故障组件。这反映了传统根因分析在 LLM 场景中的局限性：故障往往难以察觉，并与加速器绑定执行有关，GPU 降频案例便是如此。仅靠标准指标并不足够；首词元时间、令牌速率和 actor 级延迟等信号至关重要。面向微服务设计的现有根因分析工具为后续工作奠定了坚实基础，但仍需调整，以应对 LLM 服务系统的架构和运维复杂性，尤其是驱动程序问题或设备插件故障等 GPU 相关故障。

有效性威胁。一些因素可能会限制我们研究结果的普遍适用性。我们关注了 CPU 占用过高、内存泄漏和网络延迟等常见故障，并加入了 GPU 降频故障，但未涵盖漏洞或配置错误。我们调整了故障参数，以适应大语言模型环境更高的容量，这可能会影响检测难度。我们对 RCA 基准测试做了少量修改，以支持大语言模型遥测和更多方法，这可能会影响与原始评估的可比性 [38]。负载生成采用固定的大语言模型使用模式，但可能无法涵盖现实部署的全部情况。我们有意基于基础系统遥测数据（CPU、内存、网络、图形处理器）评估根因分析方法，而非采用不具备普遍适用性的专用应用程序指标；这样既与相关工作保持一致，又能最大限度地降低测量和计算开销，并避免因主观选择指标而引入偏差。尽管这些因素会影响有效性，此项评估仍为了解根因分析在大语言模型系统中的适用性提供了有价值的见解。

5.5 差距分析与可观测性指南

我们的实验表明，许多故障并不会单纯通过 CPU、内存、网络或图形处理器指标显现，而是会通过框架特定行为中的副作用暴露出来，例如 Ray actor 重启或模型加载失败。这些信号往往会被标准可观测性技术栈遗漏，或默认隐藏起来。例如，Ray 偶尔会使用基于 IP 的 Pod 直接通信，这会增加网络追踪的难度，并绕过服务网格，因此需要进行特殊配置[15]。我们还必须调整 RayCluster 放置策略

[15]https://docs.ray.io/en/latest/cluster/kubernetes/k8s-ecosystem/istio.html，访问时间：2026年3月

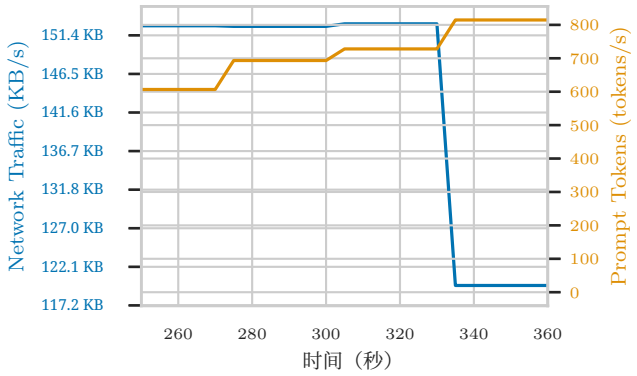

图 7：内存异常后应用特定信号的对比：网络流量下降，而每秒提示词元数上升，展现出反直觉的影响；若缺乏深入了解，便难以解读。

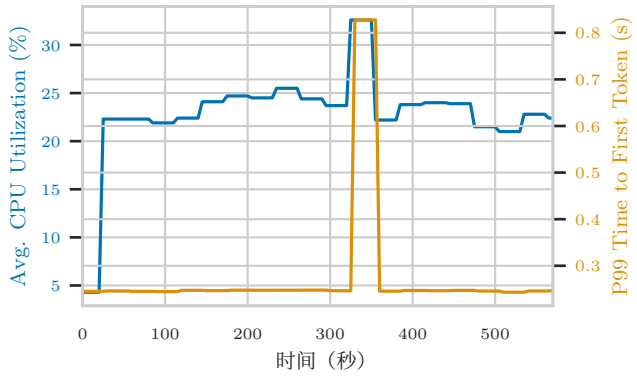

图 8：自稳定实验：注入 CPU Hogfault 会导致 CPU 利用率上升和首词元时间（TTFT）激增；根因分析识别出故障工作节点 Pod 并触发重启，从而恢复稳定性能，同时也凸显了将首词元时间（TTFT）解读为容量指标所面临的挑战。

以减少内部回环和共享内存通信，因为这些情况在开箱即用时很难观测到。

要弥合这一可观测性差距，大语言模型系统的可观测性必须覆盖多个层面：系统层（CPU、内存、图形处理器）、编排层（Pod 和容器状态），以及应用层语义（例如执行器健康状态、词元时序）。这需要在 Prometheus 中进行自定义检测埋点，访问大语言模型专属指标（如首词元时间和端到端延迟），并对进程事件进行结构化日志记录。为遥测数据补充模型名称、执行器 ID 和部署拓扑等上下文信息，对于有效的关联分析至关重要。虽然追踪可以揭示请求流，但由于框架变更迅速且受性能限制，直接进行检测埋点往往并不实际。在此，基于 eBPF 的追踪（例如使用 DeepFlow）提供了一种非侵入式替代方案，无需修改应用代码即可捕获通信模式和执行延迟。

因此，现代可观测性策略必须在低开销、深入洞察和适应性之间取得平衡。为满足大语言模型推理的需求，需要将云原生可观测性与大语言模型及框架特定知识相结合。然而，即使能够深入洞察，确定哪些遥测信号具有相关性也绝非易事；还必须权衡实际因素，例如框架变更的频率（对于应用特定遥测信号而言），以及初始信号选择和检测埋点的开销。即便能够获取此类信号，解读它们之间的相互关系也并非总是简单，因为相关性可能很复杂，也可能具有误导性。为说明这一挑战，我们在图 7 中对比了两个应用特定指标，凸显它们之间的关系远非显而易见：在 Ray 工作节点 Pod 中注入内存异常并导致内存溢出故障后，我们观察到报告的集群内网络流量有所下降。与此同时，以每秒提示词元数衡量的整体吞吐量有所上升，因为工作负载

会重新分配给其余工作节点 Pod，导致更多请求被批量处理，单个请求的延迟也随之增加，从而推高该指标。然而，如果不了解框架的内部行为，这种反直觉的

关系无论对人工操作人员还是根因分析方法而言，都难以解读。

5.6 迈向根因分析感知的人工智能基础设施

为了在大语言模型推理系统中支持及时、可靠的根因分析，可观测性必须发展为基础设施感知分析。传统根因分析假设因果关系是线性的，且组件行为稳定；然而，大语言模型工作负载依赖图形处理器密集型参与者、共享内存、分布式对象存储和协作式调度。根因分析方法必须调整，以纳入这些架构模式。例如，Ray actor 故障可能引发重试、重新调度或排队延迟；这些都是上游故障的症状。如果无法细致观察参与者生命周期和图形处理器行为，根因分析方法就可能错误归因。有效的根因分析应纳入对大语言模型专属抽象的认知，例如参与者执行、KV 缓存使用情况和模型阶段。还必须纳入图形处理器遥测数据、模型服务指标和面向客户端的 API 数据。处理降级状态和部分恢复模式至关重要。日志、指标和追踪数据必须跨系统层关联，并满足时间限制。一般而言，根因分析系统不应仅依赖通用信号，而应嵌入关于大语言模型行为的声明式知识：对健康参与者的精确定义、延迟分解以及故障传播细节。这样不仅能实现准确的故障定位，还能采取可执行的补救措施，其中可能包括自动执行恢复操作。

作为一个示例，我们重点介绍图 8 所示的自稳定实验，该实验是探索性系列实验的一部分。在 Ray 工作节点 Pod 中引入 CPU 异常后，整体 CPU 利用率急剧上升，首词元时间（TTFT）指标也随之飙升，表明推理性能下降。异常检测组件通过轻量级告警规则实现，并结合我们在早期分析中选定的一种根因分析方法，正确识别出受影响的 Pod 是可能的根因，并自动启动了

重启。在重启阶段，该 Pod 被移出 Kubernetes 的请求路由，使其余健康 Pod 的 CPU负载和 TTFT 值得以稳定。重启完成后，恢复的 Pod 便恢复了正常运行。这个示例说明，可观测性结合根因分析能够实现自动纠正措施。同时，它也凸显了信号可解释性方面的挑战：虽然系统的有效容量在恢复期间暂时下降，但这一影响并未反映在 TTFT 中，这可能是由于 vLLM 对该指标的定义和实现方式所致——它只测量已开始执行的请求，而不测量排队等待处理的请求。

一般而言，根因分析感知型可观测性必须与系统语义紧密契合。这指向一种更广泛的转变：从异常检测转向以人工智能工作负载的运行逻辑为基础的上下文驱动推理。这种契合是构建可靠且易于维护的人工智能基础设施的关键。

6 相关工作

以下将回顾相关工作的不同领域。

6.1 分布式与云原生系统中的传统根因分析

经典根因分析研究涵盖基于模型、统计方法和以图为中心的领域。基于模型的系统通过编码专家规则或依赖关系图来追踪症状传播。其准确率取决于人工维护，并会随着微服务拓扑的演变而迅速下降。统计方法将每项指标视为时间序列，根据关联分析或因果性评分对潜在故障原因进行排序。Seer [15] 和 LOUD [30]是这类方法的代表：二者都会标记使用趋势早于服务级别目标违规出现的资源。这些方法可扩展至数千项指标，但往往会将工作负载变化误判为故障。Moni-torRank [21], Microscope [28], 和 MicroRCA [52] 等基于图的根因分析方法则加入了结构上下文。它们构建服务图或指标图，并采用随机游走或中心性算法，找出与观测到的故障最为吻合的节点。图推理能够捕捉多跳故障传播，但依赖完整追踪信息。当图稀疏或不断变化时，偶尔会出现误报。总体而言，这些研究为微服务根因分析奠定了基础，通常假设工作负载是传统 Web 服务工作负载。近期方法 [19, 60]通过引入新的人工智能驱动技术来扩展经典的基于模型、统计方法和基于图的方法，但仍面临类似挑战（例如可扩展性和系统拓扑的演变）。

6.2 将大语言模型用作根因分析工具

近年来，大语言模型也被用于自动化部分事件诊断流程。微软关于在云运维中使用 GPT模型进行根因分析的研究表明，大语言模型可以直接根据事件数据生成合理的原因假设和缓解建议 [2, 59]。更结构化的智能体框架（如 RCAgent） [48] 将大语言模型推理与外部数据访问工具相结合，从而实现更准确、更自主的根因识别。一些近期工作开始探索将大语言模型本身集成到智能运维流水线 [27], 中，例如使用基于 GPT 的模型进行日志摘要、事件解释或查询路由 [61]。这些方法在减少运维人员工作量并提供语义丰富的洞察方面展现出潜力，但也面临提示敏感性、非确定性，以及难以将大语言模型输出集成到确定性工作流中的挑战 [47]。其他近期工作探索了用于根因分析的基于大语言模型的智能体和基准。Flow-of-Action [35]将由大语言模型驱动的多智能体框架与标准操作规程相结合，以指导诊断并减少幻觉。与普通 ReAct 智能体相比，该方法将故障定位准确率从 ~35 %提高到 64%，几乎翻了一番。作为对这类方法的补充，OpenRCA [54]推出了一个包含 335 个真实故障案例的公开基准（涵盖 68GB 的日志、指标和追踪数据），用于评估大语言模型的诊断能力。结果表明，当前模型只能处理最简单的案例：即使使用专用智能体，表现最佳的模型也仅解决了 ~11%的故障，凸显出基于大语言模型的根因分析仍有很大的改进空间。

6.3 迈向大语言模型推理服务中的故障处理

大语言模型推理构成了一类独特的基础设施。DeepSpeed、vLLM 和 Llama.cpp 等引擎针对内存和吞吐量进行了优化，并采用连续批处理和分页注意力等机制。这些优化带来了与性能调优和跨平台复杂性相关的新型故障模式。尽管以往的研究关注训练阶段缺陷， [14, 22], 但直到最近，推理才成为研究重点。Liu等人 [29] 首次对大语言模型推理中的缺陷开展了大规模分析，在五种引擎中发现了 929 个缺陷，并将其归纳为 28 类根因。值得注意的是，超过 35% 的问题表现为非崩溃症状，例如输出错误或程序挂起，因此更难检测和诊断。另一项近期对 HuggingFace Transformers 和 vLLM 的分析 [20] 发现，API误用是最主要的根因（占缺陷的 ~40%），这标志着问题类型从早期深度学习框架中的算法缺陷转向了接口层面的问题。该研究还发现，大多数缺陷都逃过了现有测试的检验（通常是因为测试用例不足）。这一转变推动了人们对专用监控和测试工具的兴趣 [55]。错误编码的词元或内存不足错误等故障，促使人们在大语言模型服务系统中定制健康检查和日志。总的来说，要确保大语言模型推理的可靠性，就需要结合引擎内部机制和故障模式来制定根因分析方法。

7 结论

随着大语言模型日益融入现实应用，确保其服务基础设施的可靠性变得至关重要。本文研究了最初为微服务架构开发的现有系统故障根因识别方法，在应用于现代语言模型推理系统时是否仍然有效。我们在符合最佳实践的大语言模型推理技术栈上进行受控故障注入，发现现有技术能够取得颇有希望的结果，但往往也会暴露出与大语言模型推理技术栈特性相关的不足。这些故障源于核心架构差异。大语言模型服务系统依赖专用硬件、动态批处理和共享内存，因此不再满足无状态、通信可追踪以及遥测数据统一等假设。结果是，传统根因分析工具有时会错误归因异常，或完全漏掉故障。我们确定了大语言模型推理的关键可观测性需求，包括感知硬件的

指标、内部排队与调度可见性，以及对异步执行的表征，这些或可支持采取有效的补救措施。

我们的研究结果表明，诊断生成式模型基础设施中的故障，需要采用修订后的分析方法，并扩大可观测性覆盖范围。本研究为调整故障诊断工具、满足日益复杂且对性能敏感的机器学习系统的需求，提供了初步的实证依据。未来，我们计划将实验扩展到更大规模的模型和集群，并探索多样化的模型并行配置。

参考文献

[1] Amey Agrawal、Nitin Kedia、Ashish Panwar、Jayashree Mohan、Nipun Kwatra、Bhargav S. Gulavani、Alexey Tumanov 和 Ramachandran Ramjee。2024年。使用 Sarathi-Serve 驯服大语言模型推理中的吞吐量与延迟权衡。载于 *OSDI*。USENIX 协会，117–134。[2] Toufique Ahmed、Supriyo Ghosh、Chetan Bansal、Thomas Zimmermann、Xuchao Zhang 和 Saravan Rajmohan。2023年。使用大语言模型为云事件推荐根因与缓解措施。载于 *ICSE*。IEEE，1737–1749。[3] Markus Anderljung、Joslyn Barnhart、Anton Korinek、Jade Leung、Cullen O'Keefe、Jess Whittlestone、Shahar Avin、Miles Brundage、Justin Bullock、Duncan Cass-Beggs、Ben Chang、Tantum Collins、Timothy Fist、Gillian K. Hadfield、Alan Hayes、Lewis Ho、Sara Hooker、Eric Horvitz、Noam Kolt、Jonas Schuett、Yonadav Shavit、Divya Siddarth、Robert Trager 和 Kevin Wolf。2023年。前沿人工智能监管：管理新兴风险，保障公共安全。*CoRR* abs/2307.03718（2023年）。[4] Devansh Arpit、Matthew Fernandez、Chenghao Liu、Weiran Yao、Wenzhuo Yang、Paul Josel、Shelby Heinecke、Eric Hu、Huan Wang、Stephen C. H. Hoi、Caiming Xiong、Kun Zhang 和 Juan Carlos Niebles。2023年。Salesforce CausalAI Library：用于时间序列和表格数据因果分析的快速、可扩展框架。*CoRR* abs/2301.10859（2023年）。[5] Algirdas Avizienis 和 Jean-Claude Laprie。1986年。可信计算：从概念到设计多样性。*Proc. IEEE* 74，5（1986年），629–638。[6] Algirdas Avizienis、Jean-Claude Laprie、Brian Randell 和 Carl E. Landwehr。2004年。可信与安全计算的基本概念与分类。*IEEE Trans. Dependable Secur. Comput.* 1，1（2004年），11–33。[7] Luiz André Barroso 和 Urs Hölzle。2009年。数据中心即计算机：仓储规模计算机设计入门。Morgan & Claypool 出版社。[8] Ali Basiri、Niosha Behnam、Ruud de Rooij、Lorin Hochstein、Luke Kosewski、Justin Reynolds 和 Casey Rosenthal。2016年。混沌工程。*IEEE Softw.* 33，3（2016年），35–41。[9] Betsy Beyer、Chris Jones、Jennifer Petoff 和 Niall Richard Murphy（编）。2016年。站点可靠性工程：谷歌如何运行生产系统。O'Reilly Media。[10] Rishi Bommasani、Drew A. Hudson、Ehsan Adeli、Russ B. Altman、Simran Arora、Sydney von Arx、Michael S. Bernstein、Jeannette Bohg、Antoine Bosselut、Emma Brunskill、Erik Brynjolfsson、Shyamal Buch、Dallas Card、Rodrigo Castellon、Niladri S. Chatterji、Annie S. Chen、Kathleen Creel、Jared Quincy Davis、Dorottya Demszky、Chris Donahue、Moussa Doumbouya、Esin Durmus、Stefano Ermon、John Etchemendy、Kawin Ethayarajh、Li Fei-Fei、Chelsea Finn、Trevor Gale、Lauren E. Gillespie、Karan Goel、Noah D. Goodman、Shelby Grossman、Neel Guha、Tatsunori Hashimoto、Peter Henderson、John Hewitt、Daniel E. Ho、Jenny Hong、Kyle Hsu、Jing Huang、Thomas Icard、Saahil Jain、Dan Jurafsky、Pratyusha Kalluri、Siddharth Karamcheti、Geoff Keeling、Fereshte Khani、Omar Khattab、Pang Wei Koh、Mark S. Krass、Ranjay Krishna、Rohith Kuditipudi 等。2021年。基础模型的机遇与风险。*CoRR* abs/2108.07258（2021年）。[11] Lequn Chen、Zihao Ye、Yongji Wu、Danyang Zhuo、Luis Ceze 和 Arvind Krishnamurthy。2024年。Punica：多租户 LoRA 服务。载于 *MLSys*。mlsys.org。[12] Wei-Lin Chiang、Lianmin Zheng、Ying Sheng、Anastasios Nikolas Angelopoulos、Tianle Li、Dacheng Li、Banghua Zhu、Hao Zhang、Michael I. Jordan、Joseph E. Gonzalez 和 Ion Stoica。2024年。Chatbot Arena：一个基于人类偏好评估大语言模型的开放平台。载于 *ICML*。OpenReview.net。[13] Domenico Cotroneo、Roberto Pietrantuono、Stefano Russo 和 Kishor S. Trivedi。2016年。缺陷是如何显现的？对软件缺陷显现特征的全面研究。*J. Syst. Softw.* 113（2016年），27–43。[14] Weihao Cui、Ji Zhang、Han Zhao、Chao Liu、Wenhao Zhang、Jian Sha、Quan Chen、Bingsheng He 和 Minyi Guo。2025年。XPUTimer：诊断千卡以上规模 GPU 集群中大语言模型训练的异常。*CoRR*（2025年）。[15] Yu Gan、Yanqi Zhang、Kelvin Hu、Dailun Cheng、Yuan He、Meghna Pancholi 和 Christina Delimitrou。2019年。Seer：利用大数据应对云微服务性能调试的复杂性。载于 *ASPLOS*。ACM。[16] Clive WJ Granger. 1980年. 因果关系检验：个人观点. 《经济动态与控制期刊》2 (1980年), 329–352. [17] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. 2021年. 利用异构数据源诊断微服务中的性能问题. 载于 ISPA/BDCloud/SocialCom/SustainCom. IEEE.[18] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer. 1997年. 故障注入技术与工具. 《Computer》30, 4 (1997年), 75–82.[19] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Kumar Saini, Saurabh Bagchi, and Murat Kocaoglu. 2022年. 通过因果发现分析微服务中的故障根因. 载于 NeurIPS.[20] Weipeng Jiang, Xiaoyu Zhang, Xiaofei Xie, Jiongchi Yu, Yuhan Zhi, Shiqing Ma, and Chao Shen. 2025年. 根基崩裂：对大语言模型库中错误与测试实践的全面研究. CoRR abs/2506.12320 (2025年). [21] Myunghwan Kim, Roshan Sumbaly, and Sam Shah. 2013年. 面向服务架构中的根因检测. 载于 SIGMETRICS. ACM.[22] Apostolos Kokolis, Michael Kuchnik, John Hoffman, Adithya Kumar, Parth Malani, Faye Ma, Zachary DeVito, Shubho Sengupta, Kalyan Saladi, and Carole- Jean Wu. 2025年. 重新审视大规模机器学习研究集群的可靠性. 载于 HPCA. IEEE.[23] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Z hang, and Ion Stoica. 2023年. 利用PagedAttention实现大语言模型服务的高效内存管理. 载于 SOSP. ACM.[24] Mirko Lenz, Lorik Dumani, Ralf Schenkel, and Ralph Bergmann. 2024年. ArgServices：论证机器的微服务架构. 载于 RATIO（《计算机科学讲义》第14638卷）. Springer.[25] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022年. 结合干预识别的在线服务系统因果推断根因分析. 载于 SIGKDD. ACM.[26] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, Zhekang Chen, Wenchi Zhang, Xiaohui Nie, Kaixin Sui, and Dan Pei. 2021年. 通过轨迹分析实现微服务系统的实用根因定位. 载于 IWQOS. IEEE.[27] Shiau Hong Lim and Laura Wynter. 2024年. 用于IT遥测和业务事件的可解释Transformer模型. 载于 SSE. IEEE.[28] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018年. Microscope：利用微服务环境中的因果图精准定位性能问题. 载于 ICSOC（《计算机科学讲义》第11236卷）. Springer.[29] Mugeng Liu, Siqi Zhong, Weichen Bi, Yixuan Zhang, Zhiyang Chen, Zhenpeng Chen, Xuanzhe Liu, and Yun Ma. 2025年. 初探大语言模型推理引擎中的错误. CoRR abs/2506.09713 (2025年).[30] Leonardo Mariani, Cristina Monni, Mauro Pezzè, Oliviero Riganelli, and Rui Xin. 2018年. 云系统中的故障定位. 载于 ICST. IEEE计算机学会.[31] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, and Dan Pei. 2020年. 通过因果推断定位微服务中的故障根因. 载于 IWQoS. IEEE.[32] Sergio Moreschini, Shahrzad Pour, Ivan Lanese, Daniel Balouek, Justus Bogner, Xiaozhou Li, Fabiano Pecorelli, Jacopo Soldani, Eddy Truyen, and Davide Taibi. 2025年. 微服务生命周期中的人工智能技术：系统映射研究. 《 Computing 》 107, 4 (2025 年), 100.[33] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018年. Ray：面向新兴人工智能应用的分布式框架. 载于 OSDI. USENIX 协会.[34] Subhash Nerella, Sabyasachi Bandyopadhyay, Jiaqing Zhang, Miguel Contreras, Scott Siegel, Aysegul Bumin, Brandon Silva, Jessica Sena, Benjamin Shickel, Azra Bihorac, Kia Khezeli, and Parisa Rashidi. 2024年. 医疗保健领域的Transformer与大语言模型：综述. 《人工智能医学》154 (2024年), 102900.[35] Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui Li, Gaogang Xie, and Dan Pei. 2025年. Flow-of-Action：融合标准操作流程的基于大语言模型的根因分析多智能体系统. 载于 WWW. ACM, 422–431. [36] Luan Pham, Huong Ha, and Hongyu Zhang. 2024年. BARO：通过多变量贝叶斯在线变点检测实现微服务的鲁棒根因分析. 《ACM软件工程进展》1, FSE (2024年), 2214–2237. [37] Luan Pham, Huong Ha, and Hongyu Zhang. 2024年. 基于因果推断的微服务系统根因分析：我们走了多远？. 载于 ASE. ACM.[38] Luan Pham, Hongyu Zhang, Huong Ha, Flora Salim, and Xiuzhen Zhang. 2025年. RCAEval：基于遥测数据的微服务系统根因分析基准. 载于 WWW. ACM.[39] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. 2019年. ϵ-诊断：大规模微服务平台中小窗口长尾延迟的无监督实时诊断. 载于 WWW. ACM.[40] Jiaxin Shan, Varun Gupta, Le Xu, Haiyang Shi, Jingyuan Zhang, Ning Wang, Linhui Xu, Rong Kang, Tongping Liu, Yifei Zhang, Yiqing Zhu, Shuowei Jin,

Gangmuk Lim, Binbin Chen, Zuzhi Chen, Xiao Liu, Xin Chen, Kante Yin, Chak- Pong Chung, Chenyu Jiang, Yicheng Lu, Jianjun Chen, Caixue Lin, Wu Xiang, Rui Shi, and Liguang Xie. 2025年. AIBrix：迈向可扩展且经济高效的大语言模型推理基础设施. *CoRR* abs/2504.03648 (2025年).[41] Junxian Shen, Han Zhang, Yang Xiang, Xingang Shi, Xinrui Li, Yunxi Shen, Zijian Zhang, Yongxiang Wu, Xia Yin, Jilong Wang, Mingwei Xu, Yahui Li, Jiping Yin, Jianchang Song, Zhuofeng Li, and Runjie Nie. 2023年. 网络中心化分布式追踪与DeepFlow：以零代码方式排查微服务问题. 载于 *SIGCOMM*. ACM.[42] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph Gonzalez, and Ion Stoica. 2024年. SLoRA：可扩展地服务数千个LoRA适配器. 载于 *MLSys*. mlsys.org. [43] Shohei Shimizu, Patrik O. Hoyer, Aapo Hyvärinen, and Antti J. Kerminen. 2006年. 用于因果发现的线性非高斯无环模型. 机器学习研究杂志 7 (2006年), 2003–2030.[44] Yubo Shu, Haonan Zhang, Hansu Gu, Peng Zhang, Tun Lu, Dongsheng Li, and Ning Gu. 2024年. RAH! RecSys-Assistant-Human：以人为中心的推荐框架与大语言模型智能体. *IEEE* 计算社会系统汇刊 11, 5 (2024 年), 6759–6770.[45] Peter Spirtes, Clark Glymour, and Richard Scheines. 2000年. 因果、预测与搜索，第二版. 麻省理工学院出版社.[46] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015年. 谷歌利用Borg进行大规模集群管理. 载于 *EuroSys*. ACM. [47] Arthur Vitui and Tse-Hsun Chen. 2025年. 赋能智能运维：利用大语言模型进行IT运维管理. *CoRR* abs/2501.12461 (2025年).[48] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2024年. RCAgent：利用工具增强型大语言模型和自主智能体进行云根因分析. 载于 *CIKM*. ACM.[49] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022年. 思维链提示能够激发大语言模型的推理能力. 载于 *NeurIPS*.[50] Tianjun Weng, Wanqi Yang, Guangba Yu, Pengfei Chen, Jieqi Cui, and Chuanfu Zhang. 2021年. Kmon：基于eBPF、用于微服务系统的内核内透明监控系统. 载于 *2021 IEEE/ACM*云智能国际研讨会（*CloudIntelligence*）。

[51] Thomas Wolf、Lysandre Debut、Victor Sanh、Julien Chaumond、Clement De- langue、Anthony Moi、Pierric Cistac、Tim Rault、Rémi Louf、Morgan Funtowicz、Joe Davison、Sam Shleifer、Patrick von Platen、Clara Ma、Yacine Jernite、Julien Plu、Canwen Xu、Teven Le Scao、Sylvain Gugger、Mariama Drame、Quentin Lhoest 和 Alexander M. Rush。2020年。Transformers：最先进的自然语言处理。在 *EMNLP*。计算语言学协会。[52] Li Wu、Johan Tordsson、Erik Elmroth 和 Odej Kao。2020年。MicroRCA：微服务中性能问题的根因定位。在 *NOMS*。IEEE。[53] Ruyue Xin、Peng Chen 和 Zhiming Zhao。2023年。CausalRCA：基于因果推断的微服务应用精确细粒度根因定位。*J. Syst. Softw.* 203（2023年），111724。[54] Junjielong Xu、Qinan Zhang、Zhiqing Zhong、Shilin He、Chaoyun Zhang、Qingwei Lin、Dan Pei、Pinjia He、Dongmei Zhang 和 Qi Zhang。2025年。OpenRCA：大语言模型能定位软件故障的根因吗？在 *ICLR*。OpenReview.net。[55] Ruilin Xu、Zongxuan Xie 和 Pengfei Chen。2025年。eACGM：面向机器学习系统的无插桩性能追踪与异常检测。在 *IWQoS*。IEEE，1–6。[56] Guangba Yu、Pengfei Chen、Hongyang Chen、Zijie Guan、Zicheng Huang、Linxiao Jing、Tianjun Weng、Xinmeng Sun 和 Xiaoyun Li。2021年。MicroRank：在微服务环境中通过扩展频谱分析进行端到端延迟问题定位。在 *WWW*。ACM / IW3C2。[57] Gyeong-In Yu、Joo Seong Jeong、Geon-Woo Kim、Soojeong Kim 和 Byung- Gon Chun。2022年。Orca：面向基于 Transformer 的生成模型的分布式服务系统。在*OSDI*。USENIX 协会。[58] Guangba Yu、Gou Tan、Haojia Huang、Zhenyu Zhang、Pengfei Chen、Roberto Natella 和 Zibin Zheng。2024年。人工智能系统中的故障分析与故障注入综述。*CoRR* abs/2407.00125（2024年）。[59] Xuchao Zhang、Supriyo Ghosh、Chetan Bansal、Rujia Wang、Minghua Ma、Yu Kang 和 Saravan Rajmohan。2024年。使用基于 GPT-4 的上下文学习自动确定云事件的根因。在 第 *32* 届 *ACM* 软件工程基础国际会议配套论文集，*FSE 2024*，巴西加利尼亚斯港，*2024*年*7*月*15*日至*19*日。ACM，266–277。[60] Lecheng Zheng、Zhengzhang Chen、Dongjie Wang、Chengyuan Deng、Reon Matsuoka 和 Haifeng Chen。2024年。LEMMA-RCA：用于根因分析的大型多模态多领域数据集。*CoRR* abs/2406.05375（2024年）。[61] Aoxiao Zhong、Dengyao Mo、Guiyang Liu、Jinbu Liu、Qingda Lu、Qi Zhou、Jiesh- eng Wu、Quanzheng Li 和 Qingsong Wen。2024年。LogParser-LLM：利用大语言模型推进高效日志解析。在 *KDD*。ACM。