NeSy-Edge：计算连续体中的神经符号可信自愈

Peihan Ye*、Alfreds Lapkovskis*、Alaa Saleh†、张启扬‡和 Praveen Kumar Donta*
*计算机系统与科学系，斯德哥尔摩大学，瑞典斯德哥尔摩 164 25
†应用计算中心，奥卢大学，芬兰奥卢 90014
‡计算机科学学院，北京大学，中国北京 100876 peihanye5@gmail.com,
alfreds.lapkovskis@dsv.su.se, alaa.saleh@oulu.fi, qiyangzhang@pku.edu.cn,
praveen@dsv.su.se

arXiv:2603.21145v1 [cs.DC] 22 Mar 2026

摘要——**现代人工智能服务的计算需求正日益推动执行从集中式云转向涵盖边缘和终端设备的计算连续体。然而，这些环境的规模、异构性和跨层依赖使得维持韧性变得困难。现有故障管理方法往往过于静态、零散或繁重，难以支持及时自愈，尤其是在日志含噪且边缘资源受限的情况下。为解决这些局限，本文提出 NeSy-Edge，这是一种用于计算连续体可信自愈的神经符号框架。该框架采用边缘优先设计，由资源受限的边缘节点执行本地感知和推理，仅在最终诊断阶段调用云端模型。具体而言，NeSy-Edge 将原始运行时日志转换为结构化事件表示，构建先验约束稀疏符号因果图，并将因果证据与历史故障排查知识相结合，以进行根因分析并提出恢复建议。我们在具有多个语义噪声等级的代表性 Loghub 数据集上评估了该方法，考察了解析质量、因果推理、端到端诊断以及边缘侧资源使用情况。结果表明，即使在最高噪声等级下，NeSy-Edge 仍表现稳健，根因分析准确率最高可达 75%，端到端准确率可达 65%，同时本地内存占用约为 1500 MB。**

索引词—**计算连续体、边缘智能、互联网计算、神经符号、韧性、可信性**

I. 引言

ARTIFICIAL 智能（AI）正逐渐成为日常系统和服务的一部分；这需要更多计算能力、更快的响应，以及在计算连续体（云计算）中更好的资源管理。运营成本不断上升，促使组织重新思考这些服务的计算应在哪里、以何种方式运行。近期研究表明，计算正日益分布于云平台、边缘节点和终端设备之间，形成一个满足现代人工智能应用需求的计算连续体 [1]。计算连续体旨在实现多项目标，包括改善响应时间、促进本地数据使用和提高资源灵活性 [2]，但也带来了新的系统挑战。这些环境通常规模庞大、异构且紧密互连，计算资源（往往受资源限制）、数据流、互联网连接、能源需求、冷却系统和编排机制之间存在相互依赖关系 [3]。由于这些相互依赖，一个层或设备发生故障可能会迅速蔓延至连续体的其他部分，导致服务中断。因此，对于云计算而言，韧性已成为一项基本要求，而非可有可无的功能。这些系统必须能够在发生故障时持续运行，适应不断变化的条件，限制中断，并迅速恢复，同时避免性能或效率显著下降 [4]。

在实践中，由于云计算具有高度动态且不确定的特性 [5]，要在其中实现韧性仍然面临挑战。资源可用性会随时间变化，网络质量可能波动，设备可能意外加入或离开，工作负载也可能在云、边缘和终端节点之间动态编排。文献中的一些研究 [6], [7]探索了云计算中的容错与恢复，但这些研究通常依赖静态规则、孤立监控或集中式控制，这限制了它们在高度动态条件下解释故障并协调及时恢复的能力。在资源受限的环境中，这一问题更具挑战性，因为有限的资源制约了重量级故障管理方法的使用。此外，云计算中的韧性不仅意味着替换失效组件，还意味着维持服务连续性，并作出符合系统和应用目标的恢复决策。这需要能够在运行时观测系统状态，并对故障、跨层依赖和可行的恢复操作进行推理的机制。这种能力在云计算中至关重要，因为故障可能是局部且短暂的，也可能沿着编排层、数据流和共享基础设施级联传播。因此，云计算中的有效韧性必须超越基本的容错能力，在动态运行条件下实现自适应、协同的自愈。

上述局限性凸显了云计算需要一种新颖的自愈方法。在此类环境中，韧性不仅要求进行故障检测，还要求能够解读运行时观测结果、预测可能发生的失效，并及时做出恢复决策。此外，这种解决方案不应局限于资源丰富的平台，还应能够在资源受限的边缘设备上运行，处理含噪且异构的日志，并支持跨连续体多个层级的恢复。

贡献。我们提出 NeSy-Edge，这是一种三层神经符号框架，用于云计算中的可信自愈。我们的框架通过以下方式应对上述挑战

将原始运行时日志转换为根因分析（RCA）报告和可执行的恢复决策，从而减少不必要的数据传输，支持更一致的故障诊断，并实现跨连续体的上下文感知恢复。
NeSy-Edge 三层的设计概述如下：

- 感知层： 将含噪日志转换为结构化事件表示，同时将解析操作保留在本地并保持轻量。在此过程中，原始运行时日志经过预处理后，由三级机制进行处理，包括精确的符号缓存匹配、基于检索的历史知识库语义匹配，以及针对未见模式的轻量级本地小型语言模型（SLM）回退机制。

- 推理层： 根据结构化事件流构建先验约束稀疏符号因果图。该方法结合滑动窗口事件聚合与历史因果先验，以推断稀疏因果依赖关系并抑制不太可能的关联。

- 行动层： 将因果证据与检索到的故障排查知识相结合，用于根因分析和恢复建议。此外，该层还包含一种选择性推理机制，以便在本地证据足以支持确定性决策时减少对云的依赖。

我们使用多个数据集和评估指标，在各种场景下评估所提出的框架。据我们所知，针对这一问题设定、与之高度契合的端到端基线方法仍然有限。因此，我们侧重于分层评估，将各个组件与其特定任务的基线方法进行比较。

本文其余部分安排如下：第二节回顾最新且相关的文献。第三节介绍系统模型。第四节介绍所提出的 NeSy-Edge 框架设计。第五节展示并讨论仿真环境、所用数据集、数值结果及性能分析。最后，第六节总结全文。

II. 相关工作

近来，若干研究致力于日志序列建模，以实现故障检测和根因分析，从而提升容错能力与韧性。以下讨论最新且最相关的研究工作。

近期研究将语言模型融入日志分析，以提升语义理解能力和计算机系统失效检测的准确性。例如，Hong 等人 [8] 通过多样化采样、基于规则的大语言模型触发和动态模板合并，结合小型模型的高效性与大语言模型的准确性。类似地，Zhang 等人 [9]采用基于大语言模型的词元分类，结合语义感知模块和前缀树结构，实现可扩展的模板匹配。此外，文献中还采用自纠正上下文学习、异步调度和基于争议的采样等策略，进一步增强适应性和性能 [10], [11]。此外，利用日志的语义表示提升异常检测性能，并识别失效的根因，也日益受到关注。特别是，研究人员利用 GPT 对日志模板进行语义增强，随后提取基于句子双向变换器编码器表示（BERT）的嵌入，并融合原始表示与增强后的表示。这种混合语义建模进一步结合了基于变换器的异常检测 [12]。此外，研究采用了一种无监督的基于变换器的方法 [13] ，利用 BERT 掩码语言模型检测日志参数中的异常。Yang 等人 [14] 将 FastText 嵌入、TF-IDF 加权与结合自适应损失的鲁棒主成分分析（PCA）模型相结合，以提升动态日志模式下的效率和韧性。

Zhang 等人 [15] 提出了一种检索增强生成方法，结合日志摘要、样本增强和思维链提示，直接根据原始日志诊断失效。Xu 等人 [16]将令牌高效预处理、结构化诊断提示和基于检索增强生成的推理与工具调用相结合，不仅能够识别根本原因，还能利用历史知识执行自动修复操作。Mamanpoosh 等人 [17] 针对物联网系统采用分而治之策略，通过分层摘要处理大量非结构化日志：将日志分段，使用领域特定提示进行摘要，再汇总为连贯的诊断报告。除了这些由大语言模型驱动的推理方法，Huang 等人 [18] 还将数据依赖图构建与动态日志跟踪系统相结合，该系统综合运用正则表达式和基于大语言模型的语义分析，实现实时故障定位。 [19]提出的模块化系统整合了日志收集、预处理和基于大语言模型的语义分析，以实现自动分类和异常检测。类似地， [20] 提出了一种混合多层框架，结合基于规则的分类、句子嵌入聚类、监督学习（K近邻）和大语言模型，以实现自动日志分类，并及早检测大规模系统中的关键问题。

在相关文献中，一些研究并未针对容错或韧性对基于语言模型的模型进行适配。例如，某联邦学习框架将自注意力与双向长短期记忆网络相结合，以捕捉分布式环境中的时间模式和上下文模式 [21]。类似地，文献 [22],提出了一种用于故障检测和诊断的基于深度学习的方法，该方法将日志解析与基于 LSTM 的序列建模和分层故障分类相结合，从而能够捕捉系统各层级的时间依赖关系并进行异常检测。Kodela 等人 [23] 提出了一种混合多模态架构，将日志序列的双向长短期记忆网络与基于卷积神经网络的性能指标进行融合，随后采用随机森林分类器和基于 SHAP 的可解释性，以实现可解释的根因分析。遗传算法等进化方法也被用于开发计算成本较低的高效基于图的解析器 [24]。转向自主系统，Saleh 等人 [25] 提出了一种受生物启发的架构，采用由语言模型驱动的多智能体层来执行故障检测、因果推理和自适应恢复，从而在

复杂环境。

与以往主要针对日志解析、异常检测、基于大语言模型的诊断或恢复编排等孤立阶段的研究不同，我们提出了一种集成式、边缘优先的神经符号框架，用于云计算中的可信自愈。NeSy-Edge 的特点是采用端到端的三层设计，将轻量级本地日志感知、先验约束的符号因果推理，以及因果证据引导的根因分析和恢复建议整合到单一流水线中。不同于主要依赖集中式或纯神经推理的方法，我们的框架明确融入了符号先验、选择性云端调用和资源感知的本地执行，因此更适用于嘈杂、异构且资源受限的连续体环境。

III. 系统模型

云计算连续体涵盖一个或多个 C 云数据中心 $\mathcal{C} = \{c_i\}_{i=1}^{C}$，以及$E$ 其他异构节点 $\mathcal{E} = \{e_j\}_{j=1}^{E}$，例如物联网（IoT）设备、边缘节点、雾节点等。因此，云计算连续体可以表示为图 $G_{CC} = (\mathcal{V},\ \mathcal{R})$，其中 $\mathcal{V} = \mathcal{C} \cup \mathcal{E}$ 表示节点，$\mathcal{R} \subseteq \mathcal{V} \times \mathcal{V}$ 表示它们之间的通信关系。本文重点关注资源受限的边缘节点；不过，我们的框架也可推广至 $\mathcal{E}$ 和 $\mathcal{C}$中的其他类型设备。

在每个时间步 t，每个节点 $v \in \mathcal{V}$ 都会执行自己的一组任务（或服务） $\mathcal{T}_t(v)$，并且可能发生零个或多个故障（或事件） $\mathcal{F}_t(v)$。每个故障 f 可能持续一段连续的时间步，我们将其称为事件窗口 $w_f = \{t, \ldots, t+N-1\}$，其中 N表示窗口长度。在事件窗口期间，受监控服务会生成原始日志流 $L_{raw}^{f} = \{l_1, \ldots, l_M\}$，其中 M 表示该故障产生的日志数量。

所提出的 NeSy-Edge 框架旨在确保节点 $v \in \mathcal{V}$ 能够从失效中自愈 $f \in \mathcal{F}_t(v)$。为此，该框架会执行一系列步骤，将原始日志 L_{raw}^{f} 转换为根因分析报告 $R_{final}^{f} = (r^*, a^*)$，其中 r^* 表示对 f 的诊断根因，a^* 则是建议的纠正措施。生成的根因分析报告既用于解决当前故障，也用于改进未来的根因分析报告生成。

以下部分将详细介绍所提出的框架。为便于阅读，我们省略了 f 上标在L_{raw}、R_{final}及其他变量符号中的表示。

IV. 所提出的框架

如图 1 所示，NeSy-Edge 分为三个层：感知层、推理层，以及
行动层。感知层将原始日志解析为结构化事件模板，推理层构建受先验约束的稀疏符号因果图，而行动层则结合因果证据与历史知识，生成最终根因分析报告和修复操作。主要执行步骤在图 1 中标记为1 至10 ，以供参考。

该工作流由事件驱动。当边缘节点出现异常并触发实时系统告警 1 时，行动层

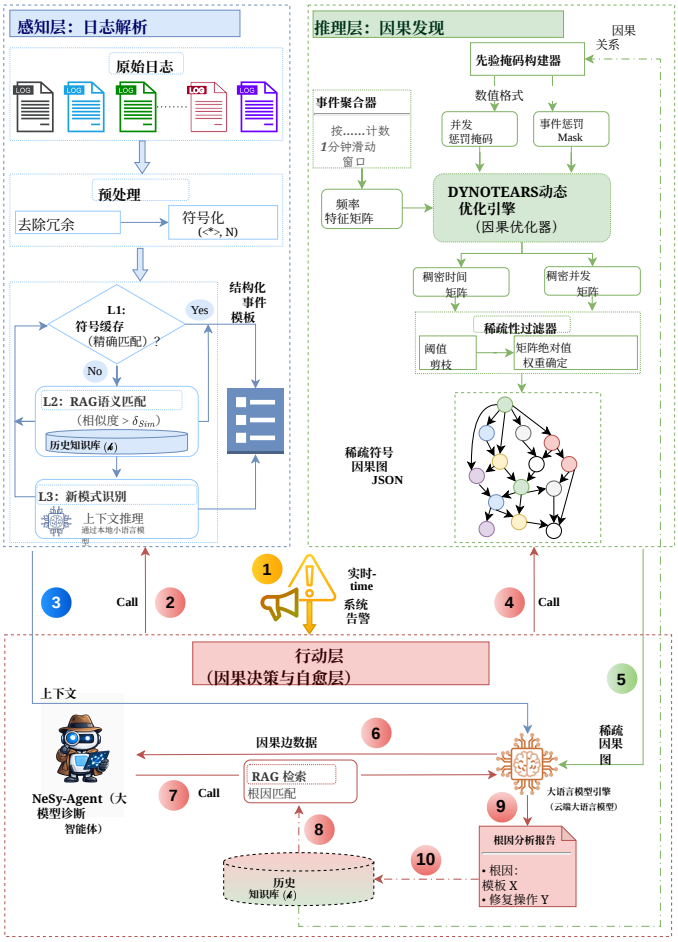

图 1。 NeSy 的整体架构和执行工作流

-边缘

会编排后续的本地感知和推理步骤，并在需要时执行最终诊断步骤，以进行根因分析和自愈。

A. 感知层

在步骤2，行动层调用感知层，将原始日志流 L_{raw} 转换为一组结构化事件模板 T_{struct}。整体解析过程总结于算法1。感知层维护一个小型本地符号缓存，用于精确复用已验证模板；共享的历史知识库 $\mathcal{K}$ 则存储更广泛的参考条目，以便进行语义检索并供后续下游复用。

首先，通过删除冗余信息并将动态字段（如 IP地址和时间戳）替换为通用占位符（例如 <*>），对每条原始日志进行预处理。原始时间戳会单独保留，以供推理层后续进行滑动窗口聚合。处理后的日志（记为 l'）随后会经过一个三级路由器，以平衡效率和泛化能力。

第一层 L1 缓存匹配会对本地符号缓存进行精确查找。如果未找到精确匹配，第二层则通过查询——

算法1： 感知层的三层日志解析路由器

感知层

输入： 原始日志流 L_{raw}、本地 L1 缓存 L1_缓存、历史知识库 $\mathcal{K}$

输出： 结构化事件模板 T_{struct}

$T_{struct} \leftarrow \emptyset$;

对于每个日志 $l \in L_{raw}$ **执行**

- $l' \leftarrow$预处理(l); ▷ 规范化动态字段
- **如果** 精确匹配(l', *L1 缓存*) **则**
  - $t \leftarrow$ 获取(l'); ▷ L1：精确缓存匹配
- **否则, 如果** 最大$_{k\in\mathcal{K}}$ 相似度($\mathbf{e}_{l'}$, $\mathbf{e}_k$) $\geq \delta_{Sim}$ **则**
  - $t \leftarrow$ 获取语义(l'); ▷ L2：检索辅助匹配
- **else**
  - 本地大语言模型提示
    - 任务 → 识别新的日志模式。
    - 执行上下文推理以抽象出模板。
  - $t \leftarrow$ 大语言模型生成(l'); ▷ L3：本地大语言模型回退
  - 更新缓存(L1 缓存 t); ▷ 立即本地更新
  - 更新知识库($\mathcal{K}$, t); ▷ 验证后
- $T_{struct} \leftarrow T_{struct} \cup \{t\}$;

返回 T_{struct};

历史知识库 $\mathcal{K}$ 并应用余弦相似度来执行检索辅助匹配。令$\mathbf{e}_{l'}$ 表示处理后的日志 l'的嵌入，令$\mathbf{e}_k$ 表示候选参考条目 $k \in \mathcal{K}$的嵌入。当最大相似度超过预定义阈值时，即接受语义匹配：

$$\max_{k\in\mathcal{K}} \mathrm{Sim}(\mathbf{e}_{l'}, \mathbf{e}_k) = \max_{k\in\mathcal{K}} \frac{\mathbf{e}_{l'}^{\top}\mathbf{e}_k}{\|\mathbf{e}_{l'}\|_2\|\mathbf{e}_k\|_2} \geq \delta_{Sim}, \quad (1)$$

其中 δ_{Sim} 是可配置的相似度阈值。如果满足这一条件，则重用匹配的参考模板；否则，将该日志视为未见模式，并传递到第三层，即本地轻量级大语言模型回退机制，通过上下文推理抽象出新模板。

为支持持续适应，本地回退机制生成的模板会立即插入 L1 缓存，以便在本地重用；而将其纳入 $\mathcal{K}$ 则在验证后异步进行。解析完成后，感知层会在步骤 3 将 T_{struct} 作为后续推理阶段的结构化上下文返回给行动层。

B. 推理层

在第 4 步，行动层调用推理层，根据结构化事件模板 T_{struct} 构建稀疏符号因果图。整体推理过程总结于算法 2 中。

首先，事件聚合器使用固定的 1分钟滑动窗口对 T_{struct}中的模板进行分组，并统计其出现次数，从而形成频率特征矩阵 $X \in \mathrm{R}^{m\times d}$，其中 m是时间窗口的数量，d 是不同事件类型的数量。因此，每个元素 $X_{u,\ j}$ 记录事件类型 j 在窗口 u中的出现次数。下文定义的 W_{mask}、A_{mask}、 W和 A也始终采用相同的事件顺序。

接下来，先验掩码构建器将历史知识库 $\mathcal{K}$ 中的历史因果知识编码为两个用于优化的惩罚掩码：(i) 用于同一时间片内依赖关系的片内惩罚掩码$W_{mask} \in \mathrm{R}^{d\times d}$，其元素计算如下：

$$W_{mask,ij} = \begin{cases} c_W^{prior}, & \text{if } (i,j) \in P_W \\ c_W^{rev}, & \text{if } (j,i) \in P_W \wedge (i,j) \notin P_W \\ c_W^{bg}, & \text{otherwise,} \end{cases} \quad (2)$$

其中， P_W 是在同一时间片内有历史依据的有向事件对；(ii) 用于相邻时间片之间滞后依赖的片间惩罚掩码 $A_{mask} \in \mathrm{R}^{d\times d}$，其元素计算如下：

$$A_{mask,ij} = \begin{cases} c_A^{prior}, & \text{if } (i,j) \in P_A \\ c_A^{rev}, & \text{if } (j,i) \in P_A \wedge (i,j) \notin P_A \\ c_A^{bg}, & \text{otherwise,} \end{cases} \quad (3)$$

其中， P_A 是在相邻时间片之间有历史依据的有向事件对。惩罚值 $c \in[0,\infty)$反映了应在多大程度上鼓励或抑制某个候选关系。具体而言，对有历史证据支持的事件-模板对赋予较小的惩罚 $c_{\cdot}^{prior}$，对未知关系赋予背景惩罚$c_{\cdot}^{bg}$，对不合理的反向关系赋予较大的惩罚 $c_{\cdot}^{rev}$。

我们通过扩展 DYNOTEARS 算法 [26]来构建因果发现模型。符号先验知识通过逐元素乘积（$\circ$）显式注入连续优化问题中：

$$\min_{W,A} \mathcal{L}(X,W,A) + \lambda_W \|W \circ W_{mask}\|_1 + \lambda_A \|A \circ A_{mask}\|_1 \quad (4)$$

$$\text{s.t. } h(W) = \mathrm{tr}(e^{W\circ W}) - d = 0, \quad (5)$$

其中， $\mathcal{L}(X,\ W,\ A)$ 是最小二乘数据拟合项， W 是同一时间片内片内依赖关系的稠密加权邻接矩阵， A 是相邻时间片之间片间依赖关系的稠密加权邻接矩阵，而 λ_W 和 λ_A 是用于控制先验惩罚强度的正则化权重。式（5）中的无环约束仅施加于 W，因为 A 中的片间边连接较早的时间片与较晚的时间片，因此不会形成同一时间片内的有向环 [26]。

公式 (4) 中的掩码项会增加不合理边的优化成本，同时保留有历史依据支持的

算法 2： 先验约束因果发现

输入： 结构化事件模板 T_{struct}，历史知识库 $\mathcal{K}$

输出： 稀疏符号因果图 G_{causal}

W_{mask}， $A_{mask} \leftarrow$ 构建先验掩码($\mathcal{K}$);

初始化稠密矩阵 W (片内), A (片间)；

优化目标函数： 求解式（4），满足式（5）；

对于每个权重 $w_{ij} \in W \cup A$，**执行**

  如果 $|w_{ij}| < \theta_{prune}$，**则**

    $w_{ij} \leftarrow 0$； ▷ 阈值剪枝

$G_{causal} \leftarrow$ 构建图(W， A);

返回 G_{causal};

算法3： NeSy-Agent诊断工作流

输入： 稀疏因果图 G_{causal}，历史故障排查知识库 $\mathcal{K}$

输出： 最终根因分析报告 $R_{final} = (r^\star, a^\star)$

NeSy-Agent 提示词

- 仅识别明确确立的因果事实-
已发布在图中。

$E_{causal} \leftarrow$ CausalNavigator(G_{causal}) ⑥;

$K_{match} \leftarrow$ 检索案例($E_{causal}, \mathcal{K}$) 7,8;

如果 确定性匹配(E_{causal}, K_{match}) **则**

  $R_{final} \leftarrow$ 局部决策(E_{causal}, K_{match});

else

  大语言模型综合

  - 输入： $E_{causal} \oplus K_{match}$
  - 决策：选择有依据的根因和修复操作。

  $R_{final} \leftarrow \mathcal{M}_{LLM}(E_{causal} \oplus K_{match})$ ⑨;

更新知识库($\mathcal{K}$， R_{final})； ▷ 之后验证

返回 R_{final} ⑩;

关系。优化后，将删除权重幅度低于剪枝阈值 θ_{prune} 的边，以提高图紧凑性。序列化之前，框架可以略微降低接近阈值的弱边权重；若剪枝后两个方向的合理性仍相近，则保留先验知识支持度较高的方向。这有助于提升图的可用性。随后，推理层会在步骤 5 将生成的稀疏符号因果图 G_{causal} 作为结构化因果证据返回给行动层，同时保留事件模板标签、有向关系和边权重，以供下游因果导航使用。

C. 行动层

行动层根据稀疏符号因果图和历史知识库生成最终根因分析报告。此阶段由*NeSy-Agent*执行：它首先从图中提取精简的因果证据，然后检索匹配的历史故障排查案例，最后确定根因和修复操作；如果仍存在歧义，则通过云端大语言模型进行判断。整体诊断流程总结于算法3中。

在第6步，*NeSy-Agent* 调用因果导航器遍历 G_{causal} 并提取有向因果证据，记为 E_{causal}。这些证据概括了与当前事件最相关的图区域，包括候选根事件、其关键上游关系以及权重最高的有向路径。该智能体不会将完整图直接传递到下游，而是将这些精简证据作为诊断的结构化依据。当需要云端推理时，智能体会使用因果约束提示，将模型限定在 E_{causal}明确支持的关系范围内，而不是引入额外的因果联系。

在第7步和8 步，智能体会对历史故障排查知识库 $\mathcal{K}$ 执行 RAG 检索，以获取匹配的故障排查知识，记为 K_{match}。检索查询由 E_{causal}中包含的事件标签和候选根事件构成，确保返回的案例与当前因果模式相匹配，而非仅与表面相似度相似。随后，将检索到的案例与E_{causal} 结合，作为最终决策的依据。

如果从图中提取的证据和检索到的案例已得出一致的诊断结果，且不存在任何歧义，*NeSy-Agent* 便会在本地输出根因分析报告，并绕过云端大语言模型。否则，在步骤 9，云端大语言模型会接收合并后的上下文 $E_{causal} \oplus K_{match}$，并通过选择一个根因及相应的修复操作来生成最终根因分析报告；这些选择均由因果证据和历史案例提供支持。最后，只有经过验证的诊断—操作对才会更新到 $\mathcal{K}$ 中，以支持未来的检索，并在步骤 10返回最终根因分析报告。

V. 结果与讨论

我们使用 Python 实现了所提出的 NeSy-Edge，并将所有方法集成到一个统一框架中。在该框架中，案例构建、噪声注入和指标计算均在单一的确定性评估流程中执行。我们的实现已发布在 GitHub 代码库中[1]。

A. 实验设置

尽管该框架是在标准工作站（配备 Apple M4、10 核中央处理器和 24GB 内存的 MacBook Pro）上评估的，但我们明确将流水线设计为模拟资源受限的边缘设备，例如典型的物联网网关。因此，我们严格隔离了本地监控路径，该路径由精确缓存、符号规则、检索

[1]https://anonymous.4open.science/r/NeSy-Edge-D794

表 I 实验配置摘要

组件	规格／配置
主机环境	Apple M4（10 核心），24 GB 内存，macOS
本地边缘模型	Qwen3-0.6B
云端智能体模型	DeepSeek-V3.2（仅限 API）
生成温度	0。0（用于诊断的严格确定性）
噪声水平	$\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$
本地执行预算	2 个 CPU 线程，2048 MB 目标内存
观测到的本地峰值常驻集大小	< 1500MB

阶段和 Qwen3-0.6B 模型 [27] 组成。该本地路径在资源受限的边缘环境中运行，并在所有通过验收的运行过程中受到监控。实际上，受监控的本地进程峰值常驻集大小（RSS）始终低于 1,500MB，表明本地执行完全处于边缘设备 2GB 的限制范围内，无需额外的计算资源。为确保框架能够在现有本地资源条件下运行，实验采用分块、可恢复的方式进行。重量级云端大语言模型 DeepSeek-V3.2 [28]，被明确排除在本地监控循环之外，仅在最终的智能体诊断阶段通过 API 调用。在实验中，我们在六个噪声水平下注入受控的语义扰动 $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$。我们根据不同系统日志各自的语义结构调整扰动策略。对于简短的控制平面事件，注入器会改写同义词元（例如，将“GET”替换为“FETCH”）。对于存储日志和异构应用日志，则会扰动解析器可见变量、锚点和状态术语。该机制模拟真实的边缘遥测歧义，并检验框架是否保持稳健，也就是在表层形式变化导致语义噪声增加时，是否仍能维持性能。其他配置细节汇总于表 I。

B. 数据集

我们的实验基于公共 Loghub 数据集集合 [29] 中的三个代表性数据集开展。这些数据集被认为涵盖了三种互补的日志特征：结构化模板、具有高度歧义的控制平面事件，以及异构长时间窗口故障。对于感知层，我们保留了一小部分不相交的干净日志作为参考数据，用于精确匹配和检索；其余案例则用于评估噪声条件下的性能。

- **HDFS：** 表示具有重复块管理模式的存储服务日志。对于感知基准测试，我们使用 160 条干净的参考日志和 2,000 个评估案例。对于推理和行动基准测试，我们分别使用 46 个基准案例和 50 个完整事件窗口。
- **OpenStack：** 包含简短的控制平面消息，这些消息在语义上相似，因此在噪声干扰下容易产生歧义。对于感知基准测试，我们使用 192 条干净的参考日志和 1600 个评估案例。对于推理和行动基准测试，我们分别使用 97 个基准案例和 50 个完整的事件窗口。
- **Hadoop：** 表示复杂且异质性更高的资源故障。与 HDFS 和 OpenStack 的日志相比，其日志更长，结构也更加多样。对于感知基准测试，我们使用 160 条干净的参考日志和 1,400 个评估案例。对于推理和行动基准测试，我们分别使用 47 个基准案例和 44 个完整事件窗口。

在所有数据集中，感知层包含 5,000 个评估案例，在三种方法和六个噪声等级下扩展为 90,000 行方法与噪声组合结果。推理层总计评估 190 个测试案例，而行动层包含 144 个完整事件窗口，由此产生约 2,600 次自主诊断运行。

C. 基线方法

为分离所提出的三层框架中各组件的贡献，我们将 NeSy-Edge 与感知层、推理层和行动层具有代表性的任务专用基线方法进行比较。

- **感知层：** 我们将其与**Drain** [30], 进行比较，后者代表传统的纯模式匹配符号解析器；同时也与**直接小语言模型**基线进行比较（通过 Qwen3- 0.6B [27]实现），该基线代表不使用符号缓存和检索路由的纯神经模板生成。
- **推理层：** 我们通过三种不同范式评估结构图质量：不依赖先验的原始**DYNOTEARS** [26]，代表连续的基于评分的优化；经典的**PC算法** [31] ，代表基于约束的条件独立性方法；以及标准的**皮尔逊** 相关图，代表朴素的统计共现。
- **行动层：** 对于下游诊断，我们将其与**基础大语言模型** [32]基线进行比较，以直接测试其在选定告警和事件上下文上的基础生成能力；同时也与**仅 RAG** [33] 基线进行比较，后者代表使用标准检索到的历史参考资料、但不包含图推导的因果证据的方法。

D. 数值结果

本节展示 NeSy-Edge 与基线方法在 HDFS、OpenStack 和 Hadoop 数据集上的数值结果，涵盖不同噪声水平下的感知层、推理层和行动层。

1）感知层评估：在这一层中，我们评估三个数据集在不同噪声水平下的解析准确率和推理延迟。

解析准确率（PA）衡量归一化后预测模板与真实模板完全匹配的告警案例所占比例。PA 的计算公式为

$$\mathrm{PA} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left[\mathrm{Norm}(\hat{t}_i) = \mathrm{Norm}(t_i)\right], \tag{6}$$

其中， N 是评估日志的数量，$\hat{}\, t_i$ 是预测的模板， t_i 是真值模板，而 Norm(·) 是

归一化函数。PA 越高，表明从含噪日志中恢复正确模板的鲁棒性越强。图2展示了在六个噪声水平下，所有三个数据集上的性能始终保持一致。与 Drain 相比，NeSy-Edge 始终取得最高的 PA，而 Drain 仅在输入格式保持稳定时表现良好。此外，直接小语言模型基线表现出中等的鲁棒性，但其精确匹配一致性低于 NeSy-Edge。这一结果表明，NeSy-Edge 的混合路由设计比单纯基于规则或单纯依赖神经解析器更可靠。

例如，在 HDFS（图2(a)）上，Drain 的表现急剧下降，从 0.95 降至 0，表明一旦原始词元结构受到严重改变，其基于树的匹配就会变得不可靠。直接小语言模型的性能下降较为缓慢，从 0.90 降至 0.39，在中等噪声范围内仍相对稳定，但在噪声水平较高时下降得更明显；相比之下，Drain 很早就出现断崖式失效。NeSy-Edge 则表现出显著更强的鲁棒性，仅从 0.99 降至 0.87。这表明，即使输入措辞发生了很大变化，结合缓存匹配、检索辅助匹配和本地大语言模型回退仍然有效。

在 OpenStack（图2(b)）上，NeSy-Edge 再次表现最佳；Drain 仍快速下降，而直接小语言模型的性能下降缓慢得多。Drain 从 1 降至 0.25，而直接小语言模型则从 0.90 降至 0.71。这种差异表明，一旦词汇锚点受到扰动，精确的基于树的匹配就会变得脆弱，而直接神经抽象对同样的扰动则相对更具容忍性。相比之下，即使在最高噪声水平下，NeSy-Edge 仍将 PA 保持在 0.93 以上。这一表现与 OpenStack 控制平面日志的结构相符：尽管存在语义噪声，许多消息仍保留着可复用的模式。

如图2(c)所示，Hadoop 的结果进一步凸显了所提设计在处理异构性更强的日志时的优势。与 HDFS 和 OpenStack 相比，Hadoop 包含更长、更多样的日志模式，这使得纯规则匹配更难实现精确恢复。因此，Drain 的起始值较低，为 0.81，随后降至 0.42；而直接小语言模型则从 0.76 降至 0.52。与在 HDFS 上相比，这两种基线方法的变化趋势更为相似且相对平缓，这表明对于更长、异构性更强的模板，精确规则匹配和直接生成都受到相同结构复杂性的制约。NeSy-Edge 依然明显优于其他方法，在噪声级别不断升高时，数值仅从 1 降至 0.91。这表明，对于结构多样的日志，NeSy-Edge 的优势不仅在于精确缓存复用，还在于能够在调用本地神经回退机制之前利用检索辅助匹配。

从这些观察结果可以看出，纯符号解析效率较高，但在语义噪声下会愈发脆弱；而纯神经解析更加灵活，但在精确恢复模板方面可靠性较低。通过结合两者，NeSy-Edge 在所有评估场景中都实现了最稳健的恢复效果。

推理延迟衡量处理单条告警所需的实际耗时

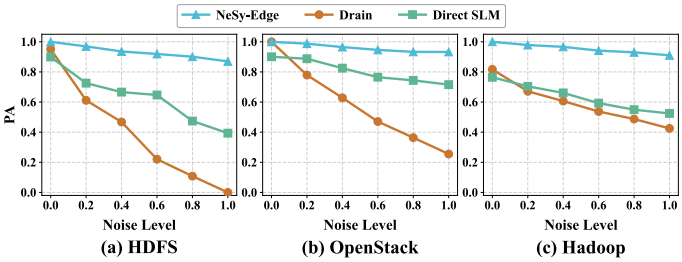

图2. 语义噪声不断增加时的解析准确率。

。在模拟的边缘预算下，低成本固然重要，但只有在输入变得困难时才增加计算量的能力也同样重要。从图 3 可以看出，Drain 在所有数据集和噪声水平下都是速度最快的方法。由于解析过程完全在本地进行且基于规则，其延迟几乎保持不变。另一极端是直接小语言模型，它是速度最慢的基线方法，因为无论输入难度如何，每条输入都会触发完整的本地生成步骤。NeSy-Edge 呈现出不同的成本特征。它不会为每条告警分配相同的计算量，而是采用一种路由策略：较简单的情况通过缓存匹配或检索辅助匹配来处理，而出现新模式的情况则交由本地大语言模型回退处理。因此，在所有设置下，其延迟都远低于直接小语言模型基线，并随着噪声水平升高而逐渐增加。

数据集层面的差异也很有参考价值。图3(a)展示了 HDFS 上的性能变化。这里，NeSy-Edge 的平均延迟从噪声为 0.0 时的 10.03 ms 增至噪声为 1.0 时的 88.55ms。在强噪声下，HDFS 中稳定输入结构的丧失更为显著，这与延迟大幅增加的情况一致，导致更多情况无法走快速匹配路径，而转入回退处理。在 OpenStack 上（见图3(b)），延迟增幅较小，从 5.41 ms 增至 54.47 ms，这表明许多带噪情况仍可在不调用本地 LLM 的情况下得到解决。如图3(3)所示，NeSy-Edge 在 Hadoop 数据集上的性能从 5.26 ms 增至 72.91ms。尽管噪声为 0.0 时的延迟与 OpenStack 接近，但在强噪声下延迟升至更高水平，这与 Hadoop 日志更大的异质性相符。

结合解析准确率结果来看，延迟结果证实了所提出的感知层设计具有实际价值。NeSy-Edge 的目标并非在原始速度上超越纯符号解析器，而是在保持强抗噪能力的同时，降低神经网络调用的成本。这正是边缘感知模块所需的行为：大多数告警通过轻量级本地匹配处理，只有在输入变得模糊不清时才会使用额外计算。

2）推理层评估：推理层用于评估不同推理方法生成的因果图的结构质量。这里，我们评估两个性能指标，即，**稀疏度**表示最终图中的边数，数值越低表示结构越紧凑、越易于解释；**平均根因**

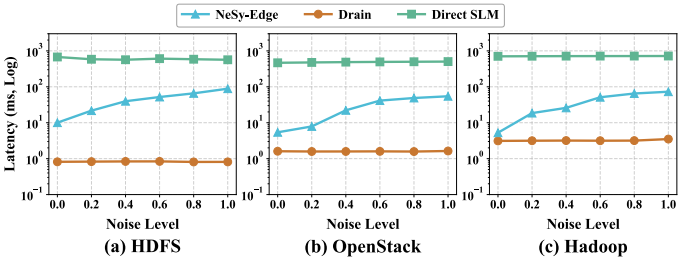

图 3. 语义噪声增加时的推理延迟。

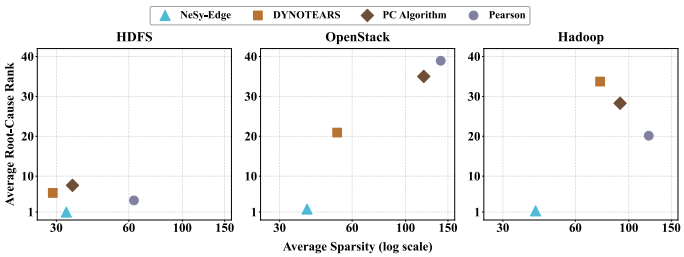

图 4. 各基线方法推理层因果图的结构质量。

排名（平均排名）用于衡量在任务对齐评估器下，真实根因关系在候选边排名中的位置。平均排名的计算方式如下：

$$\text{AvgRank} = \frac{1}{N}\sum_{i=1}^{N} r_i, \tag{7}$$

其中，M 为基准案例的数量，r_i 为案例i中分配给真实根因关系的排名位置。如果在图中未找到正确的关系，则将图中边的数量 $+1$ 赋给 r_i 作为漏检惩罚。平均排名越低且越接近 1.0，结果越好。稀疏度和平均排名应结合起来考量，因为有用的因果图既应保持紧凑，也应将正确的根因关系排在候选列表的前列。

图 4 展示了这两个目标之间的权衡。在全部三个数据集中，NeSy-Edge 始终最接近理想的左下区域，表明与基线方法相比，它在图紧凑性和根因定位之间取得了更好的平衡。皮尔逊和 PC 往往会生成更稠密或方向性较差的图，而不使用先验的 DYNOTEARS 虽然能保持相对紧凑，但仍会保留伪相关依赖。相比之下，NeSy-Edge 利用符号先验指导边选择和排序，从而生成既紧凑又有助于下游诊断的图。

详细数值见表 II。在 HDFS 上，基准数据相对集中，因此一些基线方法仍能在较好的排名位置上正确识别原因。例如，皮尔逊的平均排名达到 3.91，但其图的稀疏度高达 63.0，图也更加稠密。DYNOTEARS 在该数据集上生成的图最小，稀疏度为 29，但其平均排名仍较差，为 5.78。NeSy-Edge 将边数略微增加至 33，却将排名提升至 1.0，这表明性能提升源于保留了更有用的边集，而非单纯删除更多边。

符号先验的优势在 OpenStack 上十分明显。由于该数据集包含简短且高度相似的控制平面消息，共现很容易造成混淆。在这种情况下，皮尔逊和 PC 的图都会变得密集得多，且对正确原因的排名较低；DYNOTEARS 虽然缩小了图的规模，但正确关系在候选排名中仍远未靠前。相比之下，NeSy-Edge 既实现了最紧凑的图，也取得了最佳排名结果，稀疏度为 39，平均排名为 1.74。这表明，在 OpenStack 上，先验约束设计的价值不仅在于缩小图的规模，也在于抑制不合理依赖。

最后，Hadoop 的结果进一步证实了所提出的推理设计在更加异构的环境中的稳健性。在 Hadoop 数据集上，NeSy-Edge 实现了最佳权衡，稀疏度为 41，平均排名为 1.28。其余方法生成的图都明显更加稠密，排名结果也较差，这表明当事件空间更广、候选搜索空间更难控制时，符号先验仍然有效。

总体而言，这些结果支持推理层的设计目标。NeSy-Edge 通过将正确的根因关系置于更靠前的位置，同时保持图的紧凑性和可解释性，提升了因果图的结构实用性。这为下游行动层提供了更有力的结构化证据。

表 II 各基线方法的因果图结构质量

数据集	指标	NeSy-Edge	DYNOTEARS	PC	皮尔逊
HDFS	稀疏度	**33**	29	35	63
	平均排名	**1.00**	5.78	7.69	3.91
OpenStack	稀疏度	**39**	52	119	140
	平均排名	**1.74**	20.92	35.01	38.93
Hadoop	稀疏度	**41**	76	92	121
	平均排名	**1.28**	33.71	28.28	20.14

3）行动层评估：行动层在完整的事件窗口上评估下游诊断。**根因分析准确率**衡量生成的诊断是否正确识别了标注的根因，其计算方式为

$$\text{RCA} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left[\hat{f}_i = f_i\right], \tag{8}$$

ˆ 其中，N 为根因数量，f_i 和 f_i 分别为预测的根因标签和真实根因标签。如图 5 所示，随着语义噪声增加，所有方法的性能都会下降，但在所有数据集和噪声水平下，排名始终保持稳定：NeSy-Edge表现最佳，其次是仅 RAG，最后是基础大语言模型。基础大语言模型到仅 RAG的性能提升表明，历史参考资料提供了有用的补充证据；而 NeSy-Edge进一步取得的提升则表明，仅靠检索仍然不足

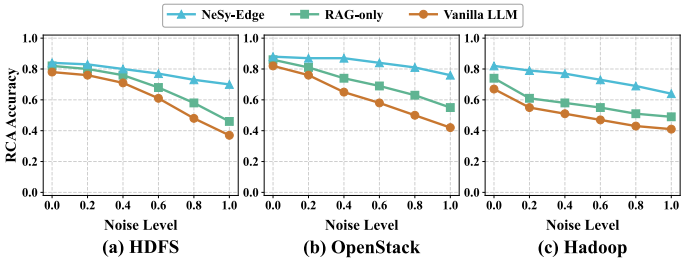

图 5. 语义噪声增加时的根因分析准确率。

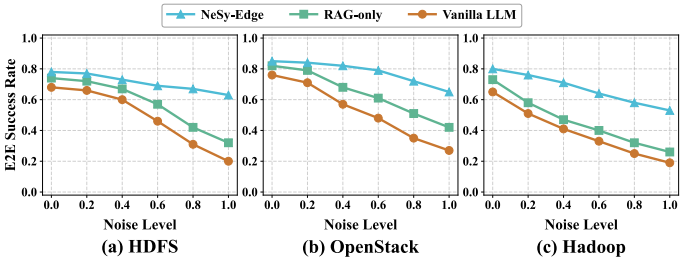

图 6. 语义噪声逐渐增加时的端到端成功率。

应对噪声条件，而且图推导的因果证据能为根因诊断带来额外且稳定的益处。

具体而言，如图 5(a) 所示，HDFS 在低噪声条件下初始准确率相对较高，但当噪声达到中高范围后，两个基线方法的准确率下降速度明显更快，而 NeSy-Edge 仍保持相对稳定，在噪声 1.0 时仍达到 0.70。由图 5(b) 可见，OpenStack 最清楚地展现了结构证据的优势。其控制平面事件通常包含较短且语义相似的消息，因此仅检索诊断仍可能受到局部噪声证据的干扰；在最大噪声下，NeSy-Edge 的根因分析准确率仍为 0.76，而仅 RAG 为 0.55，基础大语言模型为 0.42。如图 5(c) 所示，Hadoop 同样具有挑战性，因为引入噪声后所有方法都较早出现下降；即便如此，NeSy-Edge 的性能下降最为平缓，从噪声 0.0 时的 0.82 降至噪声 1.0 时的 0.64，并且始终高于两个基线方法。这些结果表明，动作层的鲁棒性始于更可靠的根因定位；对此，我们将在下文使用更严格的端到端指标进一步分析。

另一方面，**端到端成功率** 比根因分析准确率的要求更严格，因为它既要求诊断正确，也要求修复操作正确。因此，图 6 中的所有曲线都低于对应的根因分析曲线。尽管如此，在不同数据集和噪声水平下，各方法的排序始终不变，NeSy-Edge 始终优于仅 RAG 和基础大语言模型。这表明，历史参考资料和图推导的因果证据带来的益处并不止于提升诊断质量，还能转化为更可靠的端到端操作决策。

在 Hadoop 上，根因分析与端到端成功率之间的差距很能说明问题：较长的事件窗口和更严重的子类型混淆，使得将正确诊断转化为正确恢复操作变得更加困难；在噪声水平为 1.0 时，NeSy-Edge 的端到端成功率达到 0.53，而仅 RAG 为 0.26，基础大语言模型为 0.19。在 HDFS 上，两种基线方法的端到端成功率在高噪声下明显崩溃，在噪声水平为 1.0 时分别降至 0.32 和 0.20，而 NeSy-Edge 仍保持在 0.63，如图 6(a) 所示。从图 6(b) 可以看出，OpenStack 再次展现出最强的端到端鲁棒性，也最清楚地体现了将图推导的因果证据传递到下游动作决策中的收益：在最高噪声水平下，NeSy-Edge 达到 0.65，而仅 RAG 为 0.42，基础大语言模型为 0.27。同样，Hadoop 的表现也始终如一在不同噪声水平下均如此，如图 6(c) 所示。总体而言，动作层结果表明，来自上游感知和推理的结构化证据不仅能改善根因诊断，还能提升下游修复决策在噪声下的稳定性。

E. 讨论

三个层面的性能优势表明，NeSy-Edge 的优势并非来自单一组件，而是源于其神经符号设计的互补性。在感知层，NeSy-Edge 在保持延迟远低于直接本地小型语言模型的同时，对语义噪声表现出最强的鲁棒性，这说明缓存匹配、检索辅助匹配和选择性回退共同构成了一种实用的边缘侧权衡。在推理层，它生成的稀疏因果图比基线方法更有用，在提升根因排序效果的同时保留了可解释性。在行动层，这些上游优势进一步转化为更稳定的根因分析和端到端性能，表明图推导的因果证据与历史检索相辅相成，有助于下游诊断和修复动作生成。尽管实验按层分别报告，以便隔离各项贡献，但该框架以集成流水线的形式部署；行动层的提升表明，结构化感知和因果推理的改进彼此互补，并共同增强下游鲁棒性。总体而言，分层结果表明，鲁棒性从感知逐步构建，经由推理，最终落实到行动。各层持续一致的改进表明，端到端性能提升并非偶然，而是整体框架中各组件共同作用的结果。与此同时，本研究仍存在一些局限。首先，该框架主要以日志为中心，尚未纳入跟踪数据、指标或拓扑信号等多模态运行时证据，而这些证据或可进一步提升故障理解和恢复质量。其次，尽管 NeSy-Edge 遵循边缘优先设计，最终诊断阶段仍可能依赖云端推理。这些局限可以克服，也为未来的扩展指明了方向。

VI. 结论

本文提出了 NeSy-Edge，这是一种用于云计算中可信自愈的三层神经符号框架。该框架结合了轻量级结构化日志解析

（感知层）、推理层中的先验约束因果发现，以及行动层中的因果证据引导的诊断与恢复建议。我们在语义噪声逐渐增加的条件下，使用 Loghub 数据集（HDFS、OpenStack 和 Hadoop）进行了评估。NeSy-Edge 的表现始终优于分层基线方法，展现出更强的抗语义噪声能力；与直接本地小型语言模型解析相比，其边缘级延迟更低；生成的因果图具有诊断价值，且下游诊断与恢复性能更佳。尤其是在最高噪声级别下，其在 HDFS 上的解析准确率仍达 87%，在 OpenStack 上超过 93%，在 Hadoop 上约为 91%；生成的稀疏因果图分别包含 33、39 和 41 条边，平均排名接近最优，分别为 1.00、1.74 和 1.28；在资源受限的边缘节点上运行时，根因分析率最高达到 76%，端到端成功率达到 65%。这些结果表明，将符号约束与选择性神经推理相结合，能够有效实现边缘驱动的自愈，适用于噪声大且异构的环境。尽管该框架表现出色，但目前仅限于以日志为中心的自愈，尚未纳入多模态运行时证据或持续在线自适应。未来工作将通过扩展该框架，面向更广泛的故障场景实现在线自愈，并在云计算中提供更具自适应性的恢复能力，以解决这一局限。

参考文献

[1] S. Dustdar、V. C. Pujol 和 P. K. Donta，《论分布式计算连续体系统》，*IEEE* 知识与数据工程汇刊，第 35 卷，第 4 期，第 4092–4105 页，2023年。[2] B. Sedlak、V. Casamayor Pujol、A. Morichetta、P. K. Donta 和 S. Dustdar，《通过主动推理在边缘设备上进行自适应流处理》，*Evolving Systems*，第 16 卷，第 4 期，第 130 页，2025年。[3] B. Sedlak、V. C. Pujol、I. M. de Abril、P. K. Donta、A. N. Toosi 和 S. Dustdar，《计算连续体中的服务编排：结构性挑战与愿景》，2026年。 [在线]。可访问：https://arxiv.org/abs/ 2602 15794 。A. Chen、S. Ratnasamy、M. Alizadeh、M. Chowdhury、S. Guikema、R. Huang、S. Jaganathann、B. Kerkez、E. Lee、S. Low 等，《通过数字化统一构建弹性基础设施》，美国计算机协会通讯，第 69 卷，第 1 期，第 38–40 页，2026年。[4] [5] P. K. Donta、A. Lapkovskis、E. Mingozzi 和 S. Dustdar，《弹性设计——面向分布式连续体智能的主动推理》，2025年。[6] [ ] 2511 07202 在线。可访问：https://arxiv.org/abs/。A. Taghinezhad-Niar 和 J. Taheri，《边缘云连续体中面向能源和期限约束物联网工作流的容错且经济高效的调度》，*IEEE*服务计算汇刊，第 18 卷，第 5 期，第 2892–2903 页，2025年。[7] J. Aznar Poveda、S. Pedratscher、P. Burger、A. Younesi、M. Etheredge、Z. Najafabadi Samani、P. Thoman 和 T. Fahringer，《FedGuard：云边连续体中具备领导者感知能力的容错联邦学习》，载于第 *18* 届 *IEEE/ACM*国际效用计算与云计算会议论文集，丛书 UCC '25。美国纽约：美国计算机协会，2026年。[8] W. Hong、Y. Wu、L. Zhang、C. Duan、P. Xiao、M. He、X. Yang 和 Y. Li，《Cslparser：一种使用小型和大型语言模型进行日志解析的协作框架》，载于 *2025*年 *IEEE* 第 *36* 届国际软件可靠性工程研讨会（*ISSRE*），2025年，第 61–72 页。[9] C. Zhang、W. Xu、J. Liu、L. Zhang、G. Liu、J. Guan、Q. Zhou 和 S. Zhou，《SemanticLog：迈向有效且高效的大规模语义日志解析》，*IEEE*软件工程汇刊，第 52 卷，第 01 期，第 155–170 页，2026年1月。[10] Y. Wu、S. Yu 和 Y. Li，《利用大语言模型进行日志解析：采用自生成上下文学习和自我纠正》，载于*2025*年 *IEEE/ACM* 第 *33* 届国际程序理解会议，2025年，第 01–12 页。

[11] X. Chen, J. Chen, J. Shi, P. Wang 和 W. Wang，“Epas：通过异步调度大语言模型查询实现高效的在线日志解析”，载于《2025年IEEE第41届国际数据工程会议（ICDE）》，2025年，第4025–4037页。[12] W. Xu, P. Wang, H. Shi, G. Zhou, J. Yao 和 X.-Y. Zhang，“Gelog：一种用于异常检测的 GPT 增强型日志表示方法”，载于《2025年IEEE/ACM第33届国际程序理解会议》，2025年，第524–535页。[13] H. Uchida, K. Tominaga, H. Itai, Y. Li 和 Y. Nakatoh，“日志参数异常检测系统：使用含噪训练数据评估准确率”，载于《2025年IEEE国际消费电子会议（ICCE）》，2025年，第1–4页。[14] R. Yang, F. Deng, T. Liu 和 A. Han，“使用鲁棒主成分分析检测边缘设备上的日志异常”，载于《2025年第5届计算机系统国际会议》，2025年，第120–125页。[15] L. Zhang, T. Jia, M. Jia, Y. Wu, H. Liu 和 Y. Li，“Scalalog：使用大语言模型实现可扩展的基于日志的故障诊断”，载于《ICASSP 2025——2025年IEEE国际声学、语音与信号处理会议（ICASSP）》，2025年，第1–5页。[16] W. Xu, J. Luo, T. Huang, K. Sui, J. Geng, Q. Ma, I. Akasaka, X. Shi, J. Tang 和 P. Cai，“LogSage：一种经过工业验证、用于持续集成/持续交付故障检测与修复的基于大语言模型的框架”，载于《2025年第40届IEEE/ACM国际自动化软件工程会议（ASE）》，2025年，第3742–3753页。[17] A. Mamanpoosh 和 H. Karshenas，“通过对长篇文本数据进行语义分析，实现物联网系统的自动化故障诊断”，载于《2025年第9届国际物联网与应用会议（IoT）》，2025年，第1–6页。[18] X. Huang, K. Xue, L. Chen, J. Han, J. Li 和 D. S. L. Wei，“Forensix：使用大语言模型为超5G和6G网络实现自动化网络取证与诊断”，《IEEE Network》，第39卷，第5期，第74–80页，2025年。[19] X. Gao, S. Zhao, M. Pang, H. Zhang 和 Y. Zhang，“基于大语言模型的日志分析系统”，载于《2025年第11届IEEE隐私计算与数据安全国际会议（PCDS）》。美国加利福尼亚州洛斯阿拉米托斯：IEEE，2025年8月，第1–9页。[20] S. P. P、M. D. S. Ponnuru、V. H、R. Fathima、A. G. Bhosale 和 R. Harikrishnan，“一种用于及早发现关键问题并获取运营洞察的混合日志分类系统”，载于《2025年科学、工程与技术创新国际会议（ICRISET）》，2025年，第1–6页。[21] S. Xu, G. Yang 和 Q. Chen，“一种用于分布式系统日志异常检测的联邦学习框架”，载于《2025年第5届国际通信技术与信息技术会议（ICCTIT）》，2025年，第393–397页。[22] Z. Li, Y. Zhou, L. Xie, R. Cheng 和 Y. Xiao，“基于日志分析的大规模云系统自动化故障检测与诊断”，载于《2025年第2届国际智能计算与机器人会议（ICICR）》，2025年，第78–83页。[23] V. Kodela, G. Ramadugu, V. Bairy, G. Rathor 和 M. Gupta，“利用大数据分析对大规模 IT 基础设施进行自动化根因分析”，载于《2025年第2届国际计算与数据科学会议（ICCDS）》，2025年，第1–5页。[24] J. Setinsk´y 和 M. ˇZ´adn´ık，“Evoparser：一种日志解析的进化方法”，载于《NOMS 2025——2025年IEEE网络运维与管理研讨会》，2025年，第1–8页。[25] A. Saleh, P. K. Donta, R. Morabito, S. Tarkoma, A. Lindgren, Q. Zhang, S. Dustdar, S. Pirttikangas 和 L. Lov´en，“面向弹性分布式计算连续体系统的仿生智能体自愈框架”，2026 [ ] 2601 00339 。在线。可访问：https://arxiv.org/abs/ 。R. Pamfil, N. Sriwattanaworachai, S. Desai, P. Pilgerstorfer, K. Geor- gatzis, P. Beaumont 和 B. Aragam，“DYNOTEARS：从时间序列数据中进行结构学习”，载于《国际人工智能与统计会议》。PMLR，2020年，第1595–1605页。[27] A. Yang 等人，“Qwen3技术报告”，2025年。 [在线]。可访问：https://arxiv.org/abs/ 2505 09388 。DeepSeek-AI，“DeepSeek-V .2发布”，DeepSeek API文档，2025年，访问日期：2026年3月15日。 [在线]。可访问：https://api-docs.deepseek.com/news/news [26] [28] 3 251201 [29] Z. Jiang, J. Liu, J. Huang, Y. Li, Y. Huo, J. Gu, Z. Chen, J. Zhu 和 M. R. Lyu，“日志解析技术的大规模评估：进展如何？”，载于《第33届ACM SIGSOFT国际软件测试与分析研讨会论文集》，丛书：ISSTA 2024。美国纽约：ACM，2024年，第223–234页。

[30] P. He, J. Zhu, S. He, J. Li 和 M. R. Lyu，“Drain：一种采用固定深度树的在线日志解析方法”，载于*2017*年*IEEE*国际网络服务会议（*ICWS*），2017年，第33–40页。[31] P. Spirtes, C. N. Glymour 和 R. Scheines，因果、预测与搜索，第2版。麻省理工学院出版社，2000年。[32] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin,

S. Rajmohan、D. Zhang 和 T. Xu，“利用大语言模型对云端事件进行自动根因分析”，载于第十九届欧洲计算机系统会议论文集，2024年。[33] W. Xu、J. Luo、T. Huang、K. Sui、J. Geng、Q. Ma、I. Akasaka、X. Shi、J. Tang 和 P. Cai，“LogSage：一种基于大语言模型的框架，用于持续集成/持续交付故障检测与修复，并经过工业验证”，*arXiv* 预印本 *arXiv:2506.03691*，2025年。