arXiv:2603.02075v1 [cs.DC] 2 Mar 2026

Trident：异构多模态数据流水线的自适应调度

Ding Pan[1], **Zhuangzhuang Zhou**[2], **Long Qian**[2], **Binhang Yuan**[1]

[1]香港科技大学, [2]华为

摘要

大语言模型和多模态基础模型的快速普及，带来了对高质量训练数据前所未有的需求，使多模态数据准备流水线成为人工智能基础设施的关键组成部分。这些流水线将 CPU 密集型预处理与由人工智能加速器（如 GPU/NPU/TPU）支持的推理算子交错运行，并处理海量中间产物。对于这种计算范式而言，实现高吞吐量并非易事，因为多模态工作负载具有高度非平稳性：工作负载模式变化和输入相关的推理行为会导致性能迅速波动，而瞬时内存峰值可能引发内存不足（OOM）故障。现有调度器通常依赖基于阈值的自动扩缩容，或假设算子是同步且同构的，因此效率低下。

为应对这些核心挑战，我们提出了 Trident，一种面向固定资源集群上异构多模态数据流水线的自适应调度框架，通过闭环整合三个紧密耦合的层：**(i)观测层** 采用带两阶段异常过滤的高斯过程回归，为异步算子提供抗噪声的容量估计；**(ii)自适应层** 通过在线聚类跟踪工作负载模式变化，并采用内存约束贝叶斯优化，以较少样本调优算子配置，从而降低内存溢出风险；**(iii)调度层** 构建混合整数线性规划（MILP），在异构计算和带宽约束下联合优化算子并行度、放置和配置转换，并通过滚动更新权衡吞吐量提升与冷启动开销。三个层构成一个闭环控制：容量估计和配置建议输入 MILP，其决策又会触发样本失效和模型更新，从而确保系统范围内的一致性。我们在具有生产代表性的文档和视频整理流水线上对 Trident 进行了评估。与静态基线相比， Trident 使端到端吞吐量最多提升 2.01×（PDF 流水线）和 1.88×（视频处理流程），显著优于现有调度器，且开销低，适合在线重新优化。

1 引言

大语言模型（LLM）和多模态基础模型（MFM）的快速发展，带来了前所未有的高质量训练数据需求 [1–3]。处理 PB 级文档 [2, 4],、图像 [5], 和视频 [6] 的多模态数据准备流水线，已成为现代生成式人工智能开发的关键基础设施。这些流水线包含多种资源利用率各异的算子，例如视频解码、使用 CPU 的过滤算子，以及运行在 GPU、TPU 或 NPU 等人工智能加速器上的基于大语言模型的评估算子；大量中间多模态数据会在流水线中流动。

我们认为，流处理架构 [7–10] 应能为此类工作负载带来显著优势：与按阶段执行并在阶段间物化所有中间结果的批处理不同，流处理会将数据划分为大小动态变化的批次，使其在 CPU 预处理算子与 GPU 推理算子之间持续流动，并通过并发利用异构资源实现内存高效的流水线处理。面向人工智能和多模态工作负载的新兴流式数据处理系统（例如 Ray Data [10]）已展现出显著的吞吐量提升，优于 Spark 等批处理系统 [11]。

然而，要在生产环境的多模态数据流水线中实现高端到端吞吐量仍然充满挑战：首先，多模态工作负载具有高度非平稳性：数据集偏斜和输入相关的推理行为 [12, 13]会导致流水线执行期间的性能和资源利用率大幅、快速波动，其中包括加速器内存占用的瞬时峰值，可能引发内存不足（OOM）故障。因此，调度器需要对流水线性能进行建模，并依据实时性能指标做出资源分配决策，同时兼顾内存安全。其次，要准确描述由复杂异步算子构成的多模态数据流水线的性能，实际上十分困难。算子通常封装了大语言模型推理和批处理视觉模型，它们以异步方式执行，且具有复杂的内部调度逻辑、庞大的配置空间和不断变化的性能。能够提升吞吐量的配置变更（例如增大批次或延长序列）也可能使内存使用量超过设备上限，导致内存溢出引发的重启。第三，潜在最优配置的搜索空间呈指数级增长——每个算子都有独特的并行度配置，需要在流水线执行期间动态调整；此外，每个算子的并行执行器在集群中的放置位置也会影响流水线性能，因为这会带来跨服务器多模态数据传输开销和同步成本。

因此，若未针对上述挑战进行专项优化，现有流处理系统在多模态数据处理流水线中可能会遭遇严重的性能下降。例如，此类系统中的调度器要么对单个算子执行基于阈值的自动扩缩容，却不考虑端到端流水线性能；要么依赖不匹配的假设：算子是同步的、资源使用同质化，且算子间数据传输开销可以忽略不计 [10, 14–17]。

我们提出 Trident，这是一种通过三个相互集成的创新组件来应对这些挑战的调度框架。

- **观测层** 为异构多模态流水线提供抗噪性能模型。对于流水线中的每个复杂异步算子，我们不再依赖在执行重叠和动态批处理情况下会失效的同步有效时间估算器，而是使用高斯过程回归 [18]，学习从工作负载描述符到可实现吞吐量的直接映射。为确保模型反映的是可持续容量，而非暂时的利用不足，我们引入异常检测，过滤由上游饥饿或反压伪影主导的观测结果。该层可提供可靠的容量估计，从而支持有效调度，并在在线调优期间对候选配置进行有意义的评估。

- **一个自适应层**，负责在线工作负载聚类和内存约束下的配置调优。流式聚类无需离线预处理，即可识别不同的工作负载模式。为在加速器内存限制下安全地优化算子参数，该层采用内存约束贝叶斯优化 [19–21]，将加速器峰值内存视为黑盒约束，从而实现样本高效搜索，并降低探索过程中的内存溢出风险。该层会将候选配置输出至调度层，由调度层决定是否以及何时应用这些配置。

- **一个调度层**，它针对异构资源约束下的联合并行、放置和配置转换构建混合整数线性规划（MILP）。与将工作节点视为可互换资源的传统自动伸缩器不同，我们的模型将 GPU/NPU 和 CPU 视为不同的资源池，并为其设置独立的容量限制。有效源速率成为与下游容量相耦合的决策变量，从而实现吞吐量最大化，而非反应式扩缩。该模型还纳入集群拓扑和带宽约束，将带宽密集型相邻阶段协同放置，以避免跨节点传输瓶颈。MILP 纳入了控制滚动更新的整数决策变量，用于

配置转换，并权衡预期吞吐量提升与实例重启期间产生的冷启动开销。迁移惩罚项则会抑制跨优化轮次进行不必要的放置变更。

请注意，这三个层共同构成一个闭环控制系统，而非彼此独立地运行。观测层生成的容量估计为调度层中的混合整数线性规划和自适应层中的代理模型提供参数。当自适应层检测到工作负载模式变化并提出新的算子配置时，调度层会权衡预期吞吐量增益与滚动更新的冷启动开销，以决定是否采用该配置。提交配置转换后，调度层会使观测层中受影响算子的历史样本失效，从而触发模型重新初始化，确保容量估计与活跃系统状态保持一致。这种双向耦合——调度决策会反馈至观测层和自适应层，反之亦然——对于维持全系统一致性至关重要：若缺少这种耦合，过时的容量模型或未协调的配置切换就会侵蚀各层单独带来的吞吐量增益。

我们在 Ray Data 之上实现了 Trident，并在两个具有生产代表性的流水线上对其进行评估：一个文档处理流水线，结合了 PDF 解析、版面检测和基于大语言模型的 OCR；以及一个视频清理流水线，包含解码、场景检测、质量过滤和字幕生成等阶段。实验表明，Trident **(i)** 可为复杂的异步算子提供稳健、感知工作负载的容量估计；**(ii)** 能在工作负载模式变化时调整算子配置，同时缓解内存不足导致的中断；并且 **(iii)** 能在异构资源固定的情况下，通过联合优化并行度、放置和配置转换，改进端到端调度决策。总体而言，Trident 相较于静态基线，端到端吞吐量最高提升2.01×（PDF 流水线）和 1.88×（视频流水线），且开销较低，适合在线重新优化。

2 背景与动机

现代数据准备流水线越来越多地处理结合文本文档、图像和视频的多模态数据集，并反复执行以生成供下游基础模型使用的训练和评估语料库 [2, 6, 22]。这些流水线通常由多个阶段组成，例如解析、解码、过滤、转换和基于模型的推理，并且通常采用流式执行模型进行部署 [7–10], ，其中数据项会流经由算子构成的有向无环图（DAG），中间结果仅在必要时才会物化。虽然流式处理能够提高资源效率并减少物化开销，但也暴露出一些在传统同步、逐条记录处理场景中不太明显的调度挑战。

2.1 多模态流水线特征

多模态算子表现出高度依赖输入且具有非平稳性的行为。即使在单个阶段内，执行时间和可实现的吞吐量也可能随请求特征而发生显著变化，例如文档长度和布局、图像分辨率，或视频时长与场景复杂度。许多视觉和视频算子依赖加速器并以批量方式处理数据；此外，基于加速器的推理通常是异步的，并使用内部动态/连续批处理来摊销开销 [12, 13]。因此，观测吞吐量可能呈现非线性和时变特征，而短时间窗口的测量往往会受到瞬时上游饥饿、下游背压以及批处理伪影的干扰，无法反映可持续的算子容量。最后，能够提高吞吐量的配置选择（例如更大的批量大小或更长的序列限制）也可能使设备峰值内存急剧增加并触发内存溢出故障，从而使配置选择成为输入分布不断变化条件下的在线性能–安全权衡。

2.2 离线资源约束

在实践中，多模态数据准备流水线通常采用离线范式，在资源固定的异构设备集群上运行（即大量 CPU，以及数量固定的 GPU、

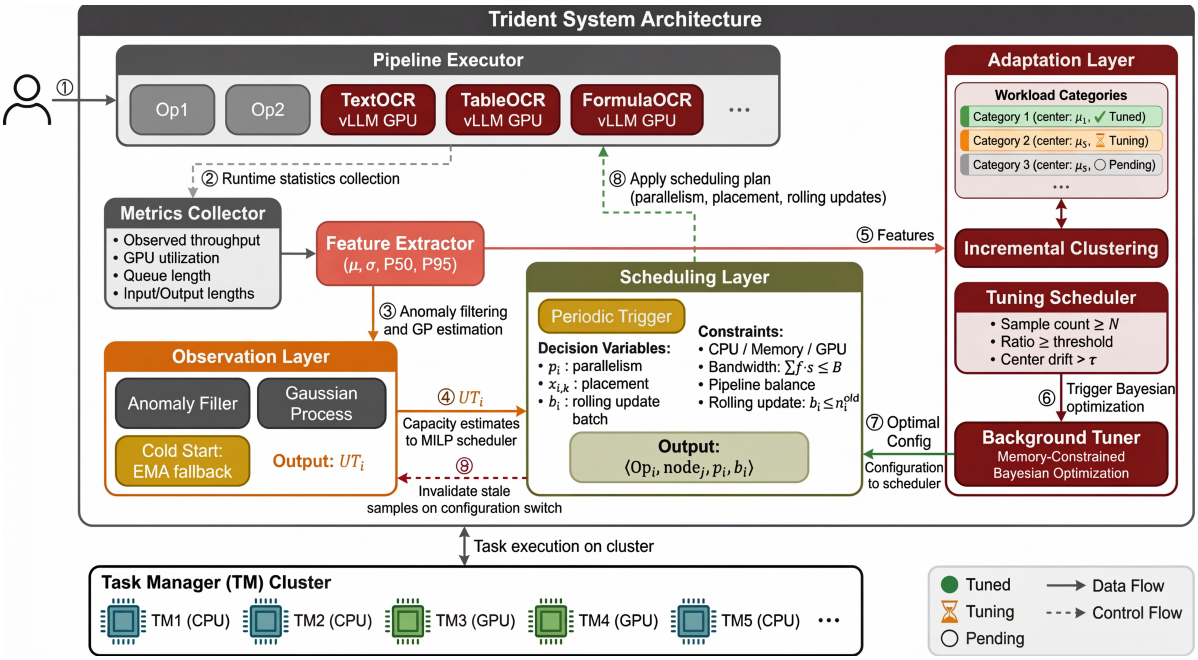

图 1 TRIDENT 系统架构。指标收集器从算子实例收集运行时统计信息，并将其送入两条并行路径：观测层过滤异常，并通过高斯过程回归估计算子吞吐量；自适应层执行工作负载聚类，并通过内存约束贝叶斯优化维护内存感知配置推荐。调度层整合容量估计与配置建议，求解一个混合整数线性规划，以联合优化并行度、放置和配置转换。在提交配置转换后，调度层向观测层发出信号，使过期样本失效（路径⑨），从而确保容量估计与当前配置保持一致。

NPU 或 TPU），并受到不可忽视的网络约束。算子具有不同的资源亲和性：有些是 CPU 密集型，有些需要加速器，而带宽密集型阶段则可能受限于大型中间数据（例如帧、裁剪区域或嵌入）的跨节点传输。因此，端到端吞吐量不仅取决于每个算子的并行度，也取决于放置决策和通信模式。与可通过自动扩缩容掩盖低效问题的弹性云环境不同，固定集群必须在严格的资源预算下，仔细协调算子并行度、放置和配置选择，同时在工作负载变化以及代价高昂的算子重启期间保持稳定。

3 系统概述

本节介绍 TRIDENT 的整体架构，并描述其组件如何交互，以实现多模态数据准备流水线的自适应调度。

3.1 系统架构

图 1 展示了 TRIDENT的架构，该架构由构建在Ray集群之上的三个集成层组成。

流水线执行器。 流水线执行器管理数据流图，实例化算子，在算子之间路由数据，并应用来自调度层的并行度更新。

指标收集器。 指标收集器从所有算子实例收集运行时统计信息，包括观测吞吐量、资源利用率、队列长度、每请求特征和峰值设备内存使用量，并将这些信息同时传递给观测层和自适应层。

观测层。 观测层通过结合

异常滤波与基于工作负载描述符的高斯过程回归 [18] 来估计每个算子的可持续吞吐量。由此得到的容量估计将输入调度层和自适应层。详情见第 4节。

自适应层。 自适应层通过在线聚类跟踪工作负载分布偏移，并通过内存约束贝叶斯优化发现最优算子配置，从而充当工作负载感知顾问：配置建议被转发至调度层，由调度层决定是否以及何时应用这些建议。详情见第5节。

调度层。 调度层构建混合整数线性规划（混合整数线性规划），在异构资源约束下联合优化算子并行度、放置和配置转换。在提交配置变更后，调度层通知观测层使过期样本失效，以确保容量估计与活跃系统状态保持一致。详情见第6节。

3.2 控制流

这三层构成一个闭环控制系统。指标收集器从算子收集观测（路径②），并将其送入两条并行路径。观测层过滤样本并生成容量估计（③→④），而自适应层执行聚类并维护配置建议（⑤→⑥→⑦）。调度层整合这两路输入以求解混合整数线性规划，从而联合确定并行度、放置和配置转换。调度决策被传播回流水线执行器（⑧）。当混合整数线性规划为某个算子选择配置变更时，调度层通过流水线执行器协调滚动更新，并向观测层发出信号以使过期样本失效（⑨），从而保持容量估计与活跃系统状态之间的一致性。

4 观测层

观测层估算每个算子的可持续吞吐量，为调度层提供容量输入。本节介绍异步算子在吞吐量估计方面面临的挑战，并提出我们结合高斯过程回归与异常检测的方法。

4.1 问题描述

对于确定性执行的同步算子，用处理的记录数除以有效时间即可得到可靠的容量指标 [15]。然而，多模态流水线中常见的异步算子——例如采用连续批处理的 大语言模型推理引擎 [12] 和面向批处理的视觉模型——并不满足这一假设：执行重叠使经过时间与已完成工作之间的对应关系变得模糊，而动态批处理则使吞吐量取决于请求到达模式，而非固定的单条记录处理成本。此外，当算子因输入不足而停滞或暂时出现积压时，原始测量值会将瞬时流水线效应与算子固有容量混为一谈。因此，观测层需要一种吞吐量估计器，能够：（i）基于工作负载特征进行条件化，以捕捉输入相关的性能变化；（ii）量化预测不确定性，支持下游决策；（iii）在非稳态观测污染模型之前将其滤除。我们通过结合高斯过程回归 [18] 与两阶段异常过滤流水线来满足这些要求。

4.2 使用高斯过程进行吞吐量建模

我们使用高斯过程（GP）回归来建模工作负载特征与算子吞吐量之间的关系 [18]。GP 非常适合这一场景，原因有二：**(i)** 无需固定函数形式即可进行灵活的非线性建模；**(ii)** 能够显式量化不确定性，这对于外推至未见过的工作负载区域以及基于模型的异常检测都很有价值。

特征表示。 对于每个异步算子，我们定义一个特征向量 $\mathbf{x}$，用于捕获与其吞吐量相关的工作负载特征。对于大语言模型推理算子，特征包括输入令牌长度的均值和标准差（μ_{in}, σ_{in}），以及输出令牌长度的均值和标准差（μ_{out}, σ_{out}）。具体而言，$\mathbf{x} = (\mu_{\text{in}}, \sigma_{\text{in}}, \mu_{\text{out}}, \sigma_{\text{out}})$。对于视觉算子，特征包括图像分辨率和批量大小。具体特征取决于算子，并在流水线定义时配置。

模型构建。我们将吞吐量 y 建模为工作负载特征的函数，具体形式如下：

$$y = f(\mathbf{x}) + \epsilon, \quad f \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{1}$$

其中，$m(\mathbf{x})$ 是均值函数，$k(\mathbf{x}, \mathbf{x}')$ 是协方差核，而 $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ 表示观测噪声。我们使用常数均值函数和 Mat´ern 5/2 核，该核具有适用于物理吞吐量关系的平滑性，同时能够稳健地应对局部变化。

增量更新。随着新的过滤后观测值到来，我们会增量更新高斯过程模型。为限制计算开销，我们使用近期观测值的滑动窗口来维护固定大小的诱导点集。当观测缓冲区超过容量时，我们会移除最旧的样本，同时确保观测到的特征空间得到充分覆盖。

吞吐量预测。给定当前的工作负载特征 $\mathbf{x}^*$，高斯过程会给出预测分布：

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \mathcal{N}(\mu^*, \sigma^{*2}) \tag{2}$$

其中，$\mathcal{D}$ 表示由过滤后的观测值构成的训练集。我们将预测均值 μ^* 作为容量估计提供给调度层。预测方差 σ^{*2} 量化了估计置信度，并支持两种用途：(i) 识别尚未探索的工作负载区域中的预测结果，以便谨慎处理；(ii) 支持基于模型的异常检测（第 4.3节）。

4.3 异常检测

并非所有吞吐量观测值都能反映算子的可持续容量。在将异常样本纳入高斯过程训练集之前，我们会通过两阶段过滤流程识别并剔除这些样本。

第1阶段：基于信号的过滤。第一阶段利用轻量且易于获取的运行时信号，检测并剔除在非稳态条件下采集的观测值。

•基于利用率的过滤。我们监测加速算子的 GPU/NPU/TPU 利用率，以及计算受限阶段的 CPU 利用率。当利用率低于阈值 τ_u 时，排除相应观测值，以避免上游饥饿导致容量估计偏低。

• 基于队列的过滤。我们通过滑动窗口跟踪队列长度，并检测两种异常模式：队列排空表明算子的处理速度超过了输入供给，而队列快速增长则表明出现了暂时性积压，从而虚增表观吞吐量。因此，我们会排除这些瞬态阶段的观测值。

第2阶段：基于模型的过滤。通过第一阶段的样本仍可能因瞬态系统效应（例如垃圾回收暂停或资源争用）而包含离群值。对于每个候选观测值（$\mathbf{x}$ y），我们查询当前高斯过程模型在 $\mathbf{x}$ 处的预测分布：

$$p(y^*|\mathbf{x}, \mathcal{D}) = \mathcal{N}(\mu, \sigma^2) \tag{3}$$

我们计算标准化残差 $z = (y - \mu)/\sigma$，并拒绝满足 $|z| > \tau_z$ 的观测，以防离群值污染后续更新。

我们想强调，这两个阶段相辅相成：基于信号的过滤能够捕捉可从运行时指标中检测到的系统性问题，而基于模型的过滤则能捕捉逃过第一阶段的偶发异常。

4.4 冷启动与样本失效

在系统启动期间或配置变更后，可能没有足够的有效样本来进行可靠的高斯过程预测。当过滤后的观测值数量低于阈值 $n_{\min}$ 时，我们会绕过高斯过程模型，仅启用基于信号的过滤（第1阶段），并将近期吞吐量观测值的指数移动平均（EMA）作为容量估计。观测数量超过 $n_{\min}$ 后，我们会切换到基于高斯过程的预测，并启用完整的两阶段过滤流程。当调度层提交

算法 1 自适应层控制流程

输入： 传入请求的特征向量 $\mathbf{x}$
确保： 更新后的集群状态和配置建议

// 阶段 1：工作负载分类
$c \leftarrow$ AssignCluster($\mathbf{x}$)
UpdateClusterStats(c, $\mathbf{x}$)
// 阶段 2：触发条件检查
如果 ShouldTriggerTuning(c)**则**
    EnqueueTuningJob(c)
结束如果
// 阶段 3：转发建议
UpdateMatchHistory(c)
$c_{dom} \leftarrow$ GetDominantCluster()
若 $c_{dom}.s =$已调优，**则**
    ForwardRecommendation(c_{dom}.配置 $c_{dom}.UT^{cand}$)
结束 if

某个算子的配置转换后，在先前配置下收集的历史样本将不再能反映该算子的行为。收到转换通知后，观测层会清空该算子的观测缓冲区，并重置其高斯过程模型，使系统恢复为基于指数移动平均的估计，直到积累足够的新样本。

5 自适应层

自适应层能够检测工作负载分布偏移，并针对每种工作负载模式识别近乎最优的算子配置。作为工作负载感知顾问，自适应层会执行在线工作负载分类和配置调优，并将建议转发给调度层；调度层则作为其统一混合整数线性规划优化的一部分，决定是否以及何时应用配置转换。算法1展示了整体控制流程。

5.1 问题描述

批量大小、最大序列长度和分块策略等算子配置会显著影响吞吐量和加速器峰值内存。最优设置取决于工作负载：较短的输入适合使用较大的批量大小，以提高设备利用率；较长的输入则需要使用较小的批量大小，以避免内存耗尽。具体的内存占用取决于框架分配器、内核融合和缓存行为，因此会表现为配置和输入分布的黑箱函数。盲目探索配置空间可能导致频繁发生内存不足（OOM）故障，进而引发工作进程重启，并使吞吐量长时间下降。因此，自适应层必须：（i）检测工作负载分布何时发生了足够大的变化，从而需要重新调优；（ii）以较高的样本效率搜索高吞吐配置，同时将内存溢出风险控制在一定范围内。我们通过在线工作负载聚类与内存约束贝叶斯优化相结合来实现这一目标。

5.2 工作负载分类

我们使用在线聚类算法维护一组工作负载类别，并在新样本到达时增量更新这些类别。

聚类表示。 每个聚类 C_i 表示为一个元组 $(\boldsymbol{\mu}_i,\ N_i,\ s_i,\ \theta_i^*)$，其中 $\boldsymbol{\mu}_i$ 表示质心，N_i 表示样本数，$s_i \in$ {待调优，调优中，已调优} 表示调优状态，θ_i^*表示可用时的最优配置。我们将聚类数量限制为 $L_{\max}$，以控制内存使用量。

特征提取。 对于每个已处理请求，我们都会提取一个低维特征向量 $\mathbf{x}$，用于表征工作负载。特征因算子而异：对于大语言模型算子，我们使用输入令牌长度和输出令牌

长度；对于视觉算子，我们使用图像分辨率和宽高比。

簇分配与更新。新样本到达时，我们会计算它与每个现有质心之间的距离。如果最小距离低于阈值 τ_d，就将该样本分配到最近的集群，并通过 $\boldsymbol{\mu}_{\text{nearest}} \leftarrow \boldsymbol{\mu}_{\text{nearest}} + \frac{1}{N_{\text{nearest}}+1}(\mathbf{x} - \boldsymbol{\mu}_{\text{nearest}})$ 增量更新质心。否则，我们会以该样本为质心创建一个新集群。如果集群数量达到上限 $L_{\max}$，则在添加新集群之前合并距离最近的两个集群。

簇维护。为适应分布漂移，我们会定期对样本计数应用指数衰减：$N_i \leftarrow \gamma \cdot N_i$ 其中 $\gamma < 1$。计数低于最低阈值的集群将被移除，使系统能够遗忘过时模式。

5.3 配置优化

TRIDENT 在线优化算子级运行时配置，以在当前主导工作负载簇下最大化可持续吞吐量。对于加速器支持的算子，激进地探索配置参数（例如批量大小、最大序列长度、分块大小）很容易触发设备内存溢出（OOM）故障，导致工作进程重启并使吞吐量长时间下降。设备峰值内存难以通过解析方式建模，因为峰值内存取决于框架分配器、内核融合和缓存行为。因此，TRIDENT 将设备内存视为黑盒约束，并在调优期间采用内存约束贝叶斯优化，从而确保推荐的配置既能实现高吞吐量，又能避免 OOM，且无需在调度层增加额外机制。

形式化地，对于可调优算子 i 和由特征向量 $\mathbf{x}_c$描述的主导工作负载簇 c，TRIDENT通过求解以下问题来选择配置 $\theta \in \Theta_i$：

$$\max_{\theta \in \Theta_i} \; UT_i(\theta; \mathbf{x}_c) \quad \text{s.t.} \quad Mem_i(\theta; \mathbf{x}_c) \le M_i^{cap} - \Delta_i, \tag{4}$$

其中，$UT_i(\theta; \mathbf{x}_c)$ 是可持续吞吐量，$Mem_i(\theta; \mathbf{x}_c)$ 是执行期间的设备峰值内存，M_i^{cap} 是该算子实例可用的设备内存容量，Δ_i 是用于考虑分配器碎片化、运行时缓存和测量噪声的安全裕度。由于吞吐量和峰值内存的评估成本都很高，且无法用闭式表达式表示，TRIDENT 会基于在线测量结果维护两个概率代理模型（在我们的实现中使用高斯过程）：

$$UT_i(\theta; \mathbf{x}_c) \sim \mathcal{N}\big(\mu_{UT}(\theta; \mathbf{x}_c),\ \sigma^2_{UT}(\theta; \mathbf{x}_c)\big)\,, \tag{5}$$

$$Mem_i(\theta; \mathbf{x}_c) \sim \mathcal{N}\big(\mu_{M}(\theta; \mathbf{x}_c),\ \sigma^2_{M}(\theta; \mathbf{x}_c)\big)\,. \tag{6}$$

每次调优评估都会记录 $(UT_i(\theta;\mathbf{x}_c)\ Mem_i(\theta;\mathbf{x}_c))$。如果评估触发内存不足，TRIDENT会将 θ标记为不可行，以便后续提议避免反复探索不安全区域。根据公式（6）中的内存代理模型，TRIDENT 会计算 θ 满足内存约束的概率：

$$PoF(\theta) = \Pr(Mem_i(\theta; \mathbf{x}_c) \le M_i^{cap} - \Delta_i) = \Phi\left(\frac{M_i^{cap} - \Delta_i - \mu_M(\theta; \mathbf{x}_c)}{\sigma_M(\theta; \mathbf{x}_c)}\right) \tag{7}$$

其中，$\Phi(\cdot)$ 是标准正态累积分布函数。

令 UT_i^+ 表示可行（未发生内存不足）的配置中观测到的最佳可持续吞吐量。TRIDENT使用吞吐量的期望改进，记为 $EI_{UT}(\theta)$，并将 $EI_{UT}(\theta)$ 与可行性相结合，定义一个约束采集函数：

$$\alpha(\theta) = EI_{UT}(\theta) \cdot PoF(\theta). \tag{8}$$

下一个配置　通过最大化 $\alpha(\theta)$ 并满足最低可行性阈值来选定。 ld:

$$\theta_{t+1} = \arg\max_{\theta \in \Theta_i} \; \alpha(\theta) \quad \text{s.t.} \quad PoF(\theta) \ge \eta, \tag{9}$$

其中 η 控制探索–安全权衡。耗尽调优预算后，TRIDENT会在满足 $PoF(\theta) \ge \eta$ 的候选项中，推荐预测吞吐量最高的最佳配置候选项，并将安全机制完全保留在调优循环内。

6 调度层

调度层会确定算子并行度和放置方式，以在固定资源约束下最大化流水线吞吐量。本节将调度问题表述为混合整数线性规划（MILP），并介绍其求解方法。

6.1 问题描述

给定一条由 n 个算子组成、在包含 K 个异构节点的集群中处理数据的流水线，调度层必须决定每个算子运行多少个实例，以及将它们放置在哪里。这个问题与传统流处理自动扩缩容有三方面不同。第一，资源是固定的，而非弹性的。离线数据准备运行在专用集群上，CPU 核心、内存和加速器设备无法按需配置。与在线流处理场景中源速率由传入数据流决定不同，在离线批处理中，源速率本身就是一个决策变量：我们的目标是在满足每个节点严格容量限制的前提下，最大化端到端吞吐量。第二，算子放置至关重要。当相邻算子位于不同节点时，中间数据必须经由网络传输。对于带宽密集型阶段——例如解压缩或帧提取之后的阶段——网络容量可能成为瓶颈。忽略放置的调度会错失将相互通信的算子部署在同一节点、减少跨节点流量的机会。第三，算子配置选择会与并行度和放置相互影响。切换到更匹配的配置有望提高稳态吞吐量，但每次转换都会因实例重启和观测预热产生冷启动开销。因此，调度层必须权衡预期收益与转换成本，并决定通过滚动更新应用配置变更是否 以及以多快的速度 进行。我们的调度层通过混合整数线性规划形式应对这些挑战，在对异构资源、数据放大、网络带宽和冷启动开销建模的同时，联合优化并行度、放置和配置转换。

6.2 系统模型

集群资源。集群由 K 个节点组成，每个节点的资源容量各不相同。节点 k 提供 C_k^{cpu} 个 CPU 核心、C_k^{mem} GB 内存和 C_k^{gpu} 个 GPU 设备。这些容量是固定的，并且在调度时已知。

流水线结构。该流水线由 n 个算子组成，采用线性数据流结构：算子 1 接收原始输入，每个算子 i 将其输出传递给算子 $i+1$，而算子 n 生成最终结果。每个算子 i 具有以下特征，这些特征由观测层和自适应层获取：

- 单位吞吐量 UT_i：单个实例的处理速率，以每秒记录数计。该值由观测层的高斯过程模型根据当前工作负载特征提供。
- 资源需求：每个实例消耗的 CPU 核心数 u_i、内存 m_i GB 和 GPU 设备数 g_i 。
- 输出数据大小 d_i^{out}：算子 i 生成的每条记录的大小，以 MB 计。
- 数据放大系数 D_i：算子 i 的输入量与原始流水线输入之比。它刻画数据在流水线中如何扩张或收缩。例如，若算子 1 将每个输入文档拆分为 10 个页面，则 $D_2 = 10$；若算子 2 过滤掉一半，则 $D_3 = 5$。

令 D_o 表示流水线输出处的放大因子。算子 i 的本地吞吐量与等效流水线输入速率之间的关系由 D_i 决定：在算子 i 处处理 D_i 条记录，对应于一条原始输入记录。

当前部署状态。对于在线重调度，我们会跟踪当前部署：$\bar{x}_{i,\,k}$ 表示当前在节点 k 上运行的算子 i 的实例数。初始启动时，$\bar{x}_{i,\,k} = 0$ 对于所有 i，k 均成立。每个算子还关联有迁移成本：启动一个新实例需要 h_i^{start} 秒，终止一个已有实例需要 h_i^{stop} 秒。

配置状态。对于每个可调优算子 i，系统都会维护其当前滚动更新状态：n_i^{old} 个实例正在运行当前配置，n_i^{new} 个实例已在之前的调度轮次中切换至候选配置。初始启动时，或没有正在进行滚动更新时，$n_i^{old} = p_i$ 且 $n_i^{new} = 0$。

自适应层提供候选配置的估算单位吞吐量 UT_i^{cand}，而观测层提供当前配置的实测单位吞吐量 UT_i^{cur}。每个实例的冷启动开销为 h_i^{cold}，包括实例重启时间以及高斯过程模型不可用期间的观测预热期。

6.3 决策变量

混合整数线性规划形式采用以下决策变量：

- $p_i \in \mathbb{Z}^+$：算子 i 的实例总数（并行度）。
- $x_{i,\ k} \in \mathbb{Z}^{\geq 0}$：部署在节点 k 上的算子 i 的实例数。
- $b_i \in \mathbb{Z}^{\geq 0}$：在此调度轮次中，从当前配置切换到候选配置的算子 i 的实例数（滚动更新批次大小）。对于不可调优的算子，或没有可用候选配置（$s_i \neq$已调优）的算子，$b_i = 0$。
- $w_{i,k,l} \in \mathbb{Z}^{\geq 0}$：从节点 k 上的算子 i 到节点 l上的算子 $i+1$ 的数据流单元。每个单元表示一个实例的输出容量。
- $\delta_{i\ k,}^{+}, \delta_{i\ ,k}^{-} \in \mathbb{Z}^{\geq 0}$：相对于当前部署，在节点 k 上增加或移除的算子 i 的实例。
- $T \geq 0$：以原始输入每秒记录数计的流水线吞吐量。
- $E_{max} \geq 0$：所有节点中的最大出口流量（MB/s）。
- $J_{mig} \geq 0$：总迁移成本（秒）。

6.4 优化目标

首要目标是最大化吞吐量 T。次要目标包括尽量降低网络峰值负载以避免带宽瓶颈，并尽量降低迁移成本以维持部署稳定性。我们通过加权求和将这些目标结合起来：

$$\max \quad T - \lambda_1 \cdot E_{max} - \lambda_2 \cdot J_{mig} \tag{10}$$

系数 λ_1 和 λ_2 是较小的正值，用于确定优先级：吞吐量优先，网络均衡和迁移成本则作为平局时的判定依据。在实践中，我们将 $\lambda_1 = 10^{-4}$ 和 $\lambda_2 = 10^{-6}$设定为相应的值，以确保只有在吞吐量相同时，次要目标才会影响解。

6.5 约束

吞吐量约束。系统吞吐量不能超过任何算子的处理能力。当算子执行滚动更新（$b_i > 0$）时，其实例会根据各自的配置状态以不同速率贡献吞吐量。在一个调度轮次开始时，算子 i 已有 n_i^{new}个实例采用候选配置，另有 n_i^{old}个实例采用当前配置。混合整数线性规划决定总并行度 p_i 和滚动批次大小 b_i。由于已完成迁移的实例不会回滚，因此按设计，$p_i \geq n_i^{new}$成立。在新部署的 p_i 个实例中：

- n_i^{new}实例以速率 UT_i^{cand}继续使用候选配置；
- b_i 实例正在切换：每个实例都会重启，并产生冷启动开销 h_i^{cold}，从而得到有效速率 $\hat{}UT_i$(公式 (11))；
- 其余 $p_i - n_i^{new} - b_i$ 个实例以速率 UT_i^{cur}保持当前配置。

迁移中的实例的有效吞吐量会计入调度窗口内的冷启动开销：

$$\hat{UT}_i = UT_i^{cand} \cdot \max\left(0,\ 1 - \frac{h_i^{cold}}{T_{sched}}\right) \tag{11}$$

由于 h_i^{cold}和 T_{sched}是已知常数，$\hat{}\ UT_i$在构建混合整数线性规划之前就已预先计算。因此，$\hat{}\ UT_i$不会引入非线性。定义如下：

$$p_i^{stay} = p_i - n_i^{new} - b_i \tag{12}$$

对于第 i个算子，吞吐量约束变为：

$$T \le \frac{D_o}{D_i} \cdot \left[p_i^{stay} \cdot UT_i^{cur} + n_i^{new} \cdot UT_i^{cand} + b_i \cdot \hat{UT}_i\right] \tag{13}$$

由于 n_i^{new}是常数（由上一轮的状态决定），所有项对于决策变量 p_i 和 b_i而言都是线性的。当没有配置转换处于活动状态或已安排时（$b_i = 0$ 和 $n_i^{new} = 0$），该约束可简化为标准约束 $T \le \frac{D_o}{D_i} \cdot p_i \cdot UT_i^{cur}$。

位置 的一致性。所有节点上的实例总数必须等于总并行度 ：

$$\sum_{k=1}^{K} x_{i,k} = p_i, \quad \forall i \in \{1, \ldots, n\} \tag{14}$$

资源约束 容量约束。每个节点的资源消耗不得超过其容量 ：

$$\sum_{i=1}^{n} u_i \cdot x_{i,k} \le C_k^{cpu}, \quad \forall k \in \{1, \ldots, K\} \tag{15}$$

$$\sum_{i=1}^{n} m_i \cdot x_{i,k} \le C_k^{mem}, \quad \forall k \in \{1, \ldots, K\} \tag{16}$$

$$\sum_{i=1}^{n} g_i \cdot x_{i,k} \le C_k^{gpu}, \quad \forall k \in \{1, \ldots, K\} \tag{17}$$

这些约束将异构资源建模为独立资源池，这不同于传统自动伸缩器将所有工作节点视为可互换、容量统一的资源。

数据流守恒。算子 i 在节点 k 上产生的数据，必须由集群中某处的算子 $i+1$ 实例消耗。流变量 $w_{i,\ k,\ l}$ 对这一数据路由进行建模：

$$\sum_{l=1}^{K} w_{i,k,l} = x_{i,k}, \quad \forall i \in \{1, \ldots, n-1\}, \forall k \tag{18}$$

$$\sum_{k=1}^{K} w_{i,k,l} = x_{i+1,l}, \quad \forall i \in \{1, \ldots, n-1\}, \forall l \tag{19}$$

第一个约束确保节点 k 上算子 i 的每个实例都将其输出发送到某处；第二个约束确保节点 l 上算子 $i+1$ 的每个实例都从某处接收输入。二者共同确保算子边界处的流量平衡。

网络带宽约束。跨节点数据传输会消耗网络带宽。对于节点 k 上的算子 i，流向其他节点的出口流量为（$x_{i,k} - w_{i,k,k}$）$\cdot UT_i \cdot d_i^{out}$MB/s，其中 $w_{i,k,k}$ 表示绕过网络的本地传输。节点 k 的总出口流量受 E_{max} 限制：

$$\sum_{i=1}^{n-1}(x_{i,k} - w_{i,k,k}) \cdot UT_i \cdot d_i^{out} \leq E_{max}, \quad \forall k \in \{1, \ldots, K\} \tag{20}$$

最小化 E_{max} 在目标函数会促使相邻算子共置，从而减少跨节点流量。

迁移核算。新部署与当前部署之间的差异决定了迁移：

$$x_{i,k} = \bar{x}_{i,k} + \delta_{i,k}^{+} - \delta_{i,k}^{-}, \quad \forall i, k \tag{21}$$

迁移总成本汇总了启动和关闭开销：

$$J_{mig} = \sum_{i=1}^{n}\sum_{k=1}^{K}\left(h_i^{start} \cdot \delta_{i,k}^{+} + h_i^{stop} \cdot \delta_{i,k}^{-}\right) \tag{22}$$

在目标函数中加入 J_{mig}，并使用系数 λ_2，会促使求解器在吞吐量相近时优先选择接近当前部署的方案，从而减少不必要的频繁变更。

RoI滚动更新约束。滚动更新过程受四项约束：

$$p_i \geq n_i^{new}, \qquad \forall i \tag{23}$$
$$b_i \leq n_i^{old}, \qquad \forall i \tag{24}$$
$$b_i \leq B_i^{max}, \qquad \forall i \tag{25}$$
$$p_i^{stay} = p_i - n_i^{new} - b_i \geq 0, \qquad \forall i \tag{26}$$

约束（23）编码了上文讨论的不可回滚语义：由于已完成转换的实例仍保留在候选配置下，求解器必须至少分配 n_i^{new}个实例。这也同时消除了吞吐量约束中的 $\min(n_i^{new}, p_i)$ 非线性。第二项和第三项分别将批量大小限制在剩余旧实例的数量和可配置的最大值以内。第四项确保留存实例数非负。只有当 $UT_i > UT_i^{cur}$ 时，混合整数线性规划才会选择 $b_i > 0$，也就是说，候选配置在扣除冷启动开销后，其吞吐量超过当前配置的吞吐量。当 UT_i^{cand}仅略有优势，或相对于 T_{sched}而言冷启动惩罚较大时，求解器会推迟转换，以避免不必要的扰动。为避免配置抖动，调度层强制实施单次转换不变量：每个算子在任何时刻至多有一个待处理的配置转换。如果算子 i 正处于转换过程中（$n_i^{old} > 0$），自适应层给出的新建议会先缓冲起来，待当前转换完成后再执行（$n_i^{old} = 0$）。因此，该形式只需跟踪每个算子的两种配置状态（当前配置和候选配置），从而保证混合整数线性规划易于求解。结合目标函数（式（10））与约束（式（12）–（26）），即可得到完整的混合整数线性规划形式，下一节将介绍其求解方法。

6.6 周期性重新调度

算法2 描述了周期性重新调度循环。在每一轮开始前，调度层会向观测层查询当前的容量估计 $\{UT_i^{cur}\}$，并向自适应层查询候选配置及其调优状态。对于每个具有就绪候选配置的可调优算子（s_i =已调优），预先通过公式（ UT_i11）计算冷启动˜折扣吞吐量。这些值用于构建混合整数线性规划，由专用的 Ray actor 异步运行求解；在返回更新后的计划之前，调度器会继续使用最新的可行解运行。

算法2 周期性重新调度

要求：重新调度间隔 T_{sched}**、MILP求解器、当前状态** $\{\bar{x}_{i,k}\}$**、滚动更新状态** $\{n_i^{old}, n_i^{new}\}$**1：循环 2：** $\{UT_i^{cur}\} \leftarrow$ **查询观测层 3：** $\{UT_i^{cand}, s_i\} \leftarrow$ **查询自适应层 4：对于每个可调优算子** i**，若其状态为** $s_i =$**已调优，则执行 5：** $\hat{}UT_i \leftarrow UT_i^{cand} \cdot$**max(0,** $1 - h_i^{cold}/T_{sched}$**) 6：结束循环 7：构建并求解混合整数线性规划** $\rightarrow$**(**p^*, x^*, b^*, w^***) 8：应用放置和配置变更 9：对于所有** i, k**，执行** $\bar{x}_{i,k} \leftarrow x^*_{i,k}$ **10：休眠** T_{sched} **11：结束循环**

求解器的输出将逐步应用：只执行当前放置与目标放置之间的差异，并根据需要平稳地启动或终止实例，以尽量减少对在途处理的干扰。对于具有 $b_i^* > 0$ 的算子，通过滚动更新使用候选配置重启指定实例，并相应更新滚动更新状态（$n_i^{new} \mathrel{+}= b_i^*$，$n_i^{old} \mathrel{-}= b_i^*$）。提交配置转换后，调度层会通知观测层，使受影响算子的历史样本失效，并触发回退到基于指数移动平均的估计（第 4.4 节），直至新配置下积累足够的新样本。

7 实现

我们将 Trident 实现为 Ray Data v2.46 的扩展。0 [10], 这是一个基于 Ray 框架构建的分布式数据处理库 [23]。

对于观测层，我们扩展了 Ray Data 的指标基础设施，以捕获每个算子的时间戳、记录数和队列统计信息。工作节点维护本地缓冲区，并按可配置的时间间隔将数据刷新到中心协调器。协调器维护滑动窗口统计信息，并通过指数平滑跟踪数据放大因子。高斯过程回归模块嵌入在调度器进程中，采用轻量级实现并进行增量更新。

自适应层维护每个算子的贝叶斯优化状态，包括针对吞吐量和设备峰值内存的代理模型。设备峰值内存从设备遥测中采集（例如 GPU 上的 NVML 或 NPU 上的厂商运行时 API），OOM 事件则从工作进程异常和重启原因中检测；这些信号会更新约束采集函数所使用的可行性模型。

对于调度层，我们用基于混合整数线性规划的优化扩展了 Ray Data 的抽象自动伸缩器接口。为避免阻塞调度循环，混合整数线性规划求解被委托给专用的 Ray actor；自动伸缩器在返回更新后的计划之前，继续按最近的可行解运行。一项关键修改是缩容策略：Ray Data 的默认策略仅终止空闲实例，而我们的实现会在指示降低并行度时将繁忙实例标记为待终止。被标记的实例会先完成当前任务再释放资源，从而在无需任务抢占的情况下实现及时伸缩。

8 评估

我们评估 Trident以回答以下研究问题：

- **RQ1**：Trident 与现有调度方法相比，在端到端吞吐量方面表现如何？
- **研究问题 2**：在相同的观测/自适应支持条件下（即受控比较），Trident 的 调度层 与现有调度器相比表现如何？
- **RQ3**：观测层的吞吐量估计有多准确？

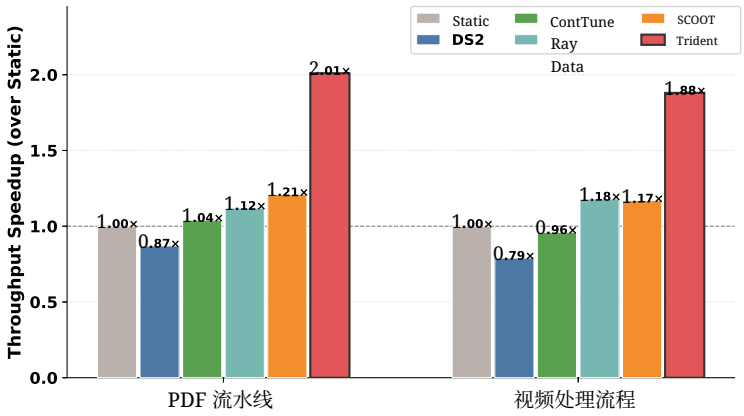

图2 PDF处理和视频整理流水线的端到端吞吐量对比。各方法在容量估算、配置调优和资源调度方面的系统级覆盖范围各不相同（表 1）。 TRIDENT 将这三个层整合在闭环中。加速比以 Static 基线为参照报告。

- **RQ4**： 自适应层能否有效检测工作负载变化、发现高吞吐配置，并在在线探索过程中保持内存安全？
- **RQ5**： 各组件对整体性能有何贡献？
- **RQ6**：系统开销是否可以接受？

8.1 实验设置

集群配置。 我们在一个由 8 台服务器组成的集群上进行实验，每台服务器均配备 8 个华为昇腾910B NPU、256 个 CPU 核心和 1 TB 内存，并通过 100Gbps 网络互连。

数据集。我们使用涵盖不同模态的两个数据集。*PDF* 数据集包含 $\sim 200k$ 份文档，分为三种类型——学术论文、年度报告和财务报告，并按类型依次处理。视频数据集包含 $\sim 410k$ 个片段，分为两类：短视频（10–30 秒， $\leq$720p）和长视频（5–10 分钟，1080p–4K），同样按类别依次处理。

流水线配置。*PDF* 整理流水线包含五个阶段（文件输入输出、解析与版面检测、区块分割、模态专属 OCR 和聚合）的 17 个算子，每份文档会扩展为 $\sim$120 个内容区块。三个基于大语言模型的 OCR 算子各需要 1 个 NPU；其余算子均为 CPU密集型。视频整理流水线包含四个阶段（基于场景的切分、美学筛选、基于 OCR 的文本筛选和基于大语言模型的字幕生成）的 9 个算子。三个 NPU 算子——基于 CLIP 的评分、CRAFT 文本检测和 Qwen2.5-VL-7B 字幕生成——各需要 1 个 NPU，其余六个算子均为 CPU密集型。

基线方法。我们将 TRIDENT 与 Static、Ray Data 的默认自动扩缩器 [10], DS2 [15], ContTune [16],以及 SCOOT [24]进行比较；基线方法的详细信息及适应方案见第8.2 节和第8.3节。

8.2 端到端系统比较 (RQ1)

我们在真实的部署条件下，将 TRIDENT 作为一个完整系统与基线方法进行评估。每种基线方法仅涵盖三个子问题中的一部分——容量估算、配置调优和资源调度；而 TRIDENT 通过其集成架构解决全部三个问题，从而实现跨组件的协调决策。表1 总结了各方法的覆盖范围。

设置。 Static 使用人工调优的固定资源分配，不进行运行时自适应。Ray Data [10] 采用默认的自动扩缩策略，根据运行中任务和利用率阈值独立扩缩每个算子。DS2 [15] 通过有效时间检测估算逐算子的处理速率，并根据数据流拓扑推导并行度，同时假设采用同步执行。ContTune [16] 在 DS2 的基础上加入保守贝叶斯

表 1 评估方法的子问题覆盖情况。

	观测	适应	调度
Static			
Ray Data			✓
DS2	✓		✓
ContTune	✓		✓
SCOOT		✓	
TRIDENT	✓	✓	✓

优化，以学习非线性的并行度—吞吐量关系，但沿用相同的有效时间检测和逐算子优化，且不具备全局资源感知。SCOOT [24] 离线使用贝叶斯优化，为每个基于大语言模型的算子调优推理引擎参数；我们为每个算子分别运行一个调优会话，并将得到的配置以静态方式部署，资源分配与 Static 相同。 TRIDENT 采用第 4–6 节所述的完整三层架构。

结果。 在图2中， TRIDENT在 PDF 和视频处理流程上分别实现了 2.01× 和 1.88× 的吞吐量，均优于所有基线方法。DS2（0.87× / 0.79×）和 ContTune（1.04× / 0.96×）依赖有效时间检测，而该检测会错误估计采用连续批处理的异步大语言模型算子的容量。由此产生的调度决策会主动错误分配资源，导致二者的表现均低于 Static，或仅勉强与其持平。Ray Data（1.12× / 1.18×）通过完全避免容量建模而优于二者——其基于阈值的响应式扩缩容不会受到不准确估计的误导，因此能够更可靠地跟踪实际需求。SCOOT（1.21× / 1.17×）是两条流水线上的最强基线方法，证实算子配置调优能够实质性地提升端到端吞吐量。然而，SCOOT 在线下孤立地调优每个算子，未考虑算子间交互；它生成无法适应运行时工作负载变化的固定配置，并且不执行跨算子资源调度。这些局限解释了 TRIDENT 仍然具备的 1.66× 和 1.61× 相对于 SCOOT 的优势： TRIDENT的观测层能够提供准确的运行时容量估计，其自适应层持续重新调优配置以跟踪工作负载漂移，而其调度层则在全局资源约束下，联合优化所有算子之间的并行度、放置和配置转换。

8.3 调度层比较（RQ2）

我们通过使所有方法的观测与适应保持不变来隔离调度层；因此，吞吐量差异仅由调度造成。

受控设置。 所有方法共享（i） TRIDENT的观测层容量估计以及（ii） TRIDENT的自适应层，后者检测工作负载模式变化并输出各算子的配置建议。由于基线方法本身不支持配置转换规划，我们为所有基线方法 配备相同的最小机制来应用这些建议：一种一次性 切换，同时重启受影响算子的所有实例（重启开销计入吞吐量）。

公平性消融。 为将滚动更新的收益与 TRIDENT的全局优化区分开来，我们加入TRIDENT (ALL-AT-ONCE)，其与 TRIDENT 完全相同，只是同样通过一次性重启来应用配置变更。

主要发现。 如表 2所示， TRIDENT即使所有方法使用完全相同的观测和配置输入，仍能实现最高吞吐量。公平性消融表明，在两种流水线上，滚动更新相较于一次性切换均带来约 5% 的适度提升，证实受控转换能够减少系统在运行状态变化期间受到的干扰。然而， TRIDENT优势的主要来源是其全局协调的感知放置调度。即使采用与基线方法相同的一次性切换语义， TRIDENT (ALL-AT-ONCE) 仍优于表现最佳的基线方法 ContTune，在 PDF 流水线上高出 1.35× ，在视频流水线上高出 1.32× ，这表明，在集群范围约束下联合优化并行度与放置，而非依赖特殊的转换机制，才是吞吐量提升的主要原因。

表 2 在相同的观测+适应输入下的调度比较。吞吐量以 Static 为基准进行归一化。

方法	**PDF**	**视频**
Static	1.00×	1.00×
Ray Data	1.22×	1.30×
DS2	1.38×	1.25×
ContTune	1.42×	1.36×
TRIDENT (一次性全部)	1.92×	1.79×
TRIDENT	**2.01**×	**1.88**×

表3 处理能力估算准确度（平均绝对百分比误差 %）。

方法	PDF	视频
真实处理速率	62.7	54.3
EMA	28.3	25.7
未进行过滤的高斯过程	24.3	21.8
高斯过程 + 信号滤波	8.4	7.1
高斯过程 + 两阶段滤波（Trident）	**5.6**	**4.8**

8.4 观测层准确性（RQ3）

我们评估观测层在端到端流水线执行期间对各算子可持续 吞吐量的估计准确度；此时，测量结果会受到上游饥饿、下游背压和瞬态失衡的影响。

设置。 我们在具有代表性的工作负载下执行两条流水线，并持续收集逐算子观测数据。将相同的样本输入每个待比较的估算器，以确保准确性差异完全归因于方法。真实容量通过以下方式获得：在与流水线相同的配置下，让每个算子在持续满负载条件下单独运行并进行性能分析。表3 报告了平均绝对百分比误差（MAPE）。

主要发现。 表 3 显示，真实处理速率估算器的误差最高（62.7%/54.3%），证实异步算子采用连续批处理时，同步有效时间假设并不成立。指数移动平均和未滤波的高斯过程分别通过平滑方差和结合工作负载特征逐步降低误差，但二者仍易受瞬态流水线效应影响。基于信号的过滤通过移除非稳态样本，大幅降低了平均绝对百分比误差，而 TRIDENT的完整两阶段滤波则通过进一步剔除高斯过程残差异常值，达到了最高准确性（5.6%/4.8%）。这些结果证实，对异步多模态算子进行准确的容量估算，既需要工作负载感知建模，也需要稳健的异常滤波。

8.5 自适应层有效性（RQ4）

我们从三个维度评估自适应层：（i）工作负载聚类准确率，（ii）配置优化效率，以及（iii）在线探索期间的 OOM 保护。

聚类准确率。 在 PDF 流水线（依次处理三种文档类型以产生状态变化）和视频处理流程（短视频片段对比长视频）上，我们将在线聚类与离线 K 均值 [25] 和 DBSCAN [26]（均可访问完整数据集）进行比较。如表 4所示，我们的在线算法无需将聚类数量作为输入，便在两条流水线上正确发现了预期的聚类数量；尽管以增量方式运行，其纯度与调整兰德指数仅略低于离线基线方法。

配置优化。我们在两个具有代表性的可调算子上评估内存约束贝叶斯优化：TextOCR（基于大语言模型，PDF 流水线）和字幕生成（Qwen2.5-VL-7B，视频处理流程）。

表4 工作负载聚类准确率。

方法	流水线	聚类	纯度	ARI
K 均值（离线）	PDF	3	0.97	0.94
DBSCAN（离线）	PDF	3	0.95	0.92
TRIDENT （在线）	PDF	3	0.95	0.89
K 均值（离线）	视频	2	0.98	0.96
DBSCAN（离线）	视频	2	0.97	0.95
TRIDENT (在线)	视频	2	0.97	0.93

表 5 配置优化对比。吞吐量以默认值 (1.0×) 归一化。 † 表示该方法将会触发内存不足的配置选为最佳配置。

方法	TextOCR (PDF)	字幕生成 (视频)
默认配置	1.00×	1.00×
随机搜索	1.18×	1.14×
网格搜索	1.22×	1.19×
无约束贝叶斯优化	1.38×†	1.35×†
有约束贝叶斯优化 (TRIDENT)	**1.36**×	**1.33**×

配置空间包含推理引擎参数，例如 max-num-seqs、max-num-batched-tokens、block-size、scheduler-delay-factor、enable-chunked-prefill 和 enable-prefix-caching，由此构成一个大型混合类型搜索空间。在持续满负载条件下，每种方法均进行 30 次评估。我们将其与随机搜索（基于 Sobol 序列 [27]）、网格搜索和无约束贝叶斯优化（不考虑内存约束的标准期望改进）进行比较。所有贝叶斯优化变体均使用 Mat´ern 5/2 核，并以 5 次随机评估初始化。如表5所示，受约束贝叶斯优化的吞吐量仅比无约束贝叶斯优化低 1–2%，同时显著优于随机搜索和网格搜索，这证实内存可行性约束带来的吞吐量权衡微乎其微。值得注意的是，无约束贝叶斯优化得出的最佳配置（以 † 标记）在持续执行流水线时会触发内存不足，从而抵消其名义上的优势。

OOM 保护。为量化端到端影响，我们运行两条流水线，在超参数和调优预算相同的情况下，对比有约束贝叶斯优化（$\eta = 0.6$， $\Delta_i = 2048$MB）与无约束贝叶斯优化。如表 6所示，有约束贝叶斯优化将 OOM 事件减少了 79–82%，累计停机时间从 462 秒降至 102 秒（PDF），从 352 秒降至 68 秒（视频）。剩余事件发生在初始随机探索阶段，此时内存代理模型的观测数据尚不充足。尽管选择的配置名义上较为保守，但由于避免了重启开销，有约束贝叶斯优化的有效吞吐量仍比无约束贝叶斯优化高 5.5%（PDF）和 4.5%（视频），与无 OOM 理想基准相比仅有 ∼3% 的差距。

8.6 消融研究 (RQ5)

我们量化了 TRIDENT 各组件对端到端性能的贡献。

设置。我们在相同的工作负载轨迹和集群配置下运行 **PDF** 和 **视频** 流水线，具体设置与第8.1节相同。我们报告归一化吞吐量，其中完整 TRIDENT = *100%*。

变体。我们评估五种变体：**(i)** TRIDENT *(完整)*；**(ii)** 无观测层，将 TRIDENT 基于高斯过程的可持续吞吐量估计替换为基于有效时间的真实处理速率估计器；**(iii)**无自适应层，禁用在线工作负载聚类和配置调优，使算子使用固定的初始配置；**(iv)** 无感知放置的调度，通过移除网络/通信建模和共置激励，使调度优化与网络无关，同时保持全局并行度和配置转换优化不变；以及**(v)** 无滚动更新，通过同时重启受影响算子的所有实例来应用每项配置变更。

结果。图3 表明，观测层是最关键的组件：移除观测层会使吞吐量降至**66.5%** （PDF）和**60.9%** （视频），这说明，准确估算可持续容量

表 6 端到端流水线执行期间的 OOM 事件及其对吞吐量的影响。

	PDF 流水线		视频处理流程	
指标	无约束	有约束	无约束	约束。
OOM 事件	14	3	11	2
累计停机时间（秒）	462	102	352	68
相对于预言机的吞吐量损失	8.7%	3.2%	7.2%	2.7%

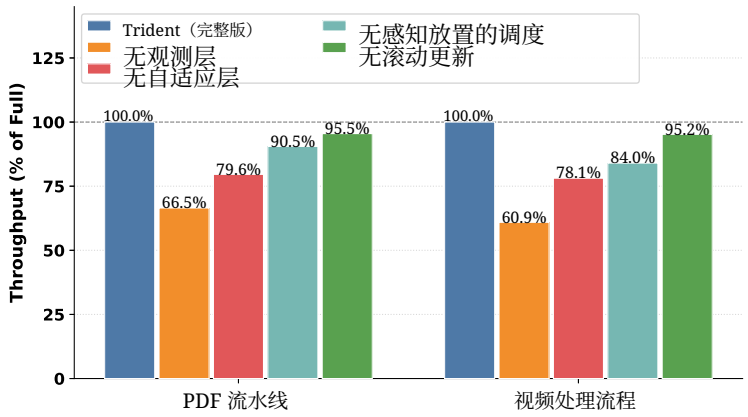

图 3 消融研究。吞吐量以完整的 TRIDENT 系统（100%）为基准进行归一化。移除任一层都会降低两条流水线的性能，其中观测层在两种情况下的影响都更大。

对于包含异步和动态批处理算子的流水线实现正确的端到端资源分配至关重要。禁用自适应层会使吞吐量降至**79.6%**（PDF）和**78.1%**（视频），这表明在线工作负载感知配置调优所带来的收益远超单纯调度。移除放置感知会导致较小但持续的性能下降，吞吐量分别降至**90.5%**（PDF）和**84.0%**（视频），这凸显了协同放置通信密集型阶段的重要性，尤其对于视频处理流程而言。最后，以**一次性**重启取代滚动更新，会使吞吐量降至**95.5%**（PDF）和**95.2%**（视频），证实受控转换能够缓解配置切换期间的冷启动中断。

8.7 系统开销（RQ6）

观测层和自适应层在 Ray Data 的调度器循环中同步执行，每次调用分别仅增加 2 毫秒和 4 毫秒的开销——与调度器自身约为~400毫秒的执行时间相比可以忽略不计。调度层的混合整数线性规划由专用 Ray actor 异步求解，因此在计算新可行解的同时，流水线会继续使用最近的可行解运行。在由 8 个节点组成的生产集群上，混合整数线性规划的平均求解时间分别为 206 毫秒（PDF 流水线）和 62 毫秒（视频处理流程）。将集群扩展到 16 个节点后，求解时间分别增加到 1,521 毫秒和 259 毫秒，仍远低于数分钟的重新调度间隔，并且完全不在关键路径上。

9 相关工作

数据流调度与流处理自动扩缩容。 诸如 Flink 等数据流系统 [9]，SparkStreaming [7]，以及 Ray Data [10] 通过反压或利用率阈值调控执行。自动扩缩容策略涵盖从基于规则的策略 [7, 14, 17, 28–32]，到基于模型的方法（如 DS2 [15] 和 Turbine [33]），这些方法根据有效时间测量推导并行度，再到基于学习的技术（如 Dragster [34]和 ContTune [16]），这些技术应用贝叶斯优化。这些方法一律假设弹性资源、同步算子以及可忽略的传输成本。值得注意的是，有效时间检测会系统性地错误估计连续批处理下异步算子的容量（第 8.4 节）。TRIDENT 通过联合优化算子并行度、设备放置和配置来弥补这些不足

在固定的异构资源预算下，通过统一的混合整数线性规划形式完成转换。

使用高斯过程进行性能建模。 高斯过程回归被广泛用于黑盒性能建模：OtterTune [35]学习数据库管理系统的性能曲面以进行参数调优，CherryPick [36] 将基于高斯过程的贝叶斯优化应用于云虚拟机选择，而 Fu 等人 [37] 表明，固有的性能波动是预测误差的主要来源，因此需要采用鲁棒观测过滤。包括 SkyGP [38] 和 DAO- GP [39] 在内的在线扩展通过局部专家维护和漂移检测来处理非平稳性，但它们都未针对包含异步算子的数据管道。 TRIDENT的观测层在此基础上，将高斯过程回归与两阶段异常过滤器相结合，移除资源饥饿和反压伪影，从而得到可持续容量估计，并随着工作负载的演变进行增量更新。

有约束的配置调优与安全探索。 受约束的贝叶斯优化已广泛应用于数据库调优：ResTune [40] 使用独立高斯过程同时对目标和服务级别协议约束进行建模，OnlineTune [41] 将在线数据库的探索限制在由高斯过程推导出的安全区域内，LlamaTune [42]通过随机嵌入降低维度，而 QTune [43] 引入查询感知编码，以提高工作负载漂移下的鲁棒性。在大语言模型推理领域，SCOOT [24] 使用带有学习得到的隐式约束的贝叶斯优化，按算子调优服务参数。 TRIDENT的自适应层采用相同的受约束贝叶斯优化公式，但目标是加速器峰值内存——违反该约束会触发内存溢出故障和长时间重启——并将配置搜索与在线工作负载模式发现相结合，将各模式推荐方案转发给调度层的混合整数线性规划求解器，以做出感知转换的承诺决策。

10 结论

本文介绍了 TRIDENT，这是一种面向运行于固定资源集群上的异构多模态数据准备管道的自适应调度框架。 TRIDENT 解决了现有流处理调度器应用于现代多模态工作负载时的关键局限，包括对异步算子的吞吐量估计不准确、面对工作负载异构性时算子配置静态不变，以及缺乏具备放置感知能力的资源分配。 TRIDENT 集成了三个紧密耦合的组件：**(i)** 一个能够抵御噪声的观测层，用于估计可持续的算子吞吐量；**(ii)** 一个自适应层，用于在线工作负载聚类和内存约束下的配置调优；以及**(iii)** 一个调度层，通过混合整数线性规划联合优化算子并行度、放置和配置转换，并且仅在预期吞吐量增益超过冷启动开销时才有选择地应用滚动更新。我们基于 Ray Data 实现了 TRIDENT ，并在具有生产代表性的文档和视频处理管道上对 TRIDENT 进行了评估。实验结果表明，与最先进的方法相比， TRIDENT可将端到端吞吐量提升至多 2.01× ，同时保持较低开销，并在工作负载模式变化时表现稳定。

参考文献

[1] Jordan Hoffmann、Sebastian Borgeaud、Arthur Mensch、Elena Buchatskaya、Trevor Cai、Eliza Rutherford、Diego de Las Casas、Lisa Anne Hendricks、Johannes Welbl、Aidan Clark、Tom Hennigan、Eric Noland、Katie Millican、George van den Driessche、Bogdan Damoc、Aurelia Guy、Simon Osindero、Karen Simonyan、Erich Elsen、Oriol Vinyals、Jack W. Rae 和 Laurent Sifre。《训练计算量最优的大语言模型》。载于 *Proceedings of the 36th International Conference on Neural Information Processing Systems* 22, 2022，NIPS '，美国纽约州 Red Hook，。Curran Associates Inc.[2] Shuai Bai、Yuxuan Cai、Ruizhe Chen、Keqin Chen、Xionghui Chen、Zesen Cheng、Lianghao Deng、Wei Ding、Chang Gao、Chunjiang Ge、Wenbin Ge、Zhifang Guo、Qidong Huang、Jie Huang、Fei Huang、Binyuan Hui、Shutong Jiang、Zhaohai Li、Mingsheng Li、Mei Li、Kaixin Li、Zicheng Lin、Junyang Lin、Xuejing Liu、Jiawei Liu、Chenglong Liu、Yang Liu、Dayiheng Liu、Shixuan Liu、Dunjie Lu、Ruilin Luo、Chenxu Lv、Rui Men、Lingchen Meng、Xuancheng Ren、Xingzhang Ren、Sibo Song、Yuchong Sun、Jun Tang、Jianhong Tu、Jianqiang Wan、Peng Wang、Pengfei Wang、Qiuyue Wang、Yuxuan Wang、Tianbao Xie、Yiheng Xu、Haiyang Xu、Jin Xu、Zhibo Yang、Mingkun Yang、Jianxin Yang、An Yang、Bowen Yu、Fei Zhang、Hang Zhang、Xi Zhang、Bo Zheng、Humen Zhong、Jingren Zhou、Fan Zhou、Jing Zhou、Yuanzhi Zhu 和 Ke Zhu。《通义千问3-VL技术报告》，2025年。

[3] Pablo Villalobos、Anson Ho、Jaime Sevilla、Tamay Besiroglu、Lennart Heim 和 Marius Hobbhahn。《观点：我们会耗尽数据吗？基于人工生成数据的大语言模型扩展限制》。载于 *Forty-first International Conference on Machine Learning* 2024，。

[4] Maurice Weber、Daniel Fu、Quentin Anthony、Yonatan Oren、Shane Adams、Anton Alexandrov、Xiaozhong Lyu、Huu Nguyen、Xiaozhe Yao、Virginia Adams、Ben Athiwaratkun、Rahul Chalamala、Kezhen Chen、Max Ryabinin、Tri Dao、Percy Liang、Christopher R´e、Irina Rish 和 Ce Zhang。RedPajama：用于训练大语言模型的开放数据集，2024年。[5] Christoph Schuhmann、Romain Beaumont、Richard Vencu、Cade Gordon、Ross Wightman、Mehdi Cherti、Theo Coombes、Aarush Katta、Clayton Mullis、Mitchell Wortsman 等。LAION-5B：用于训练新一代图文模型的开放式大规模数据集。 *Advances in neural information processing systems* 35 25278 25294, ，：–。[6] 2022 Zangwei Zheng、Xiangyu Peng、Tianji Yang、Chenhui Shen、Shenggui Li、Hongxin Liu、Yukun Zhou、Tianyi Li 和 Yang You。Open-Sora：让所有人都能高效制作视频，2024年。[7] Matei Zaharia、Tathagata Das、Haoyuan Li、Timothy Hunter、Scott Shenker 和 Ion Stoica。离散化流：大规模容错流式计算。载于 *Proceedings of the twenty-fourth ACM symposium on operating systems principles* 423 438, 2013，第–页。

[8] Tyler Akidau、Robert Bradshaw、Craig Chambers、Slava Chernyak、Rafael J Fern´andez-Moctezuma、Reuven Lax、Sam McVeety、Daniel Mills、Frances Perry、Eric Schmidt 等。数据流模型：在海量、无界、乱序数据处理中平衡正确性、延迟与成本的实用方法。 *Proceedings of the VLDB Endowment* 8 12 1792 1803, 2015，(): –。

[9] Paris Carbone、Asterios Katsifodimos、Stephan Ewen、Volker Markl、Seif Haridi 和 Kostas Tzoumas。Apache Flink：在单一引擎中进行流批处理。 *The Bulletin of the Technical Committee on Data Engineering*，38 4 2015( )，。[10] Frank Sifei Luan、Ron Yifeng Wang、Yile Gu、Ziming Mao、Charlotte Lin、Amog Kamsetty、Hao Chen、Cheng Su、Balaji Veeramani、Scott Lee 等。用于高效且容错的异构执行的流式批处理模型。 *arXiv preprint arXiv:2501.12407*，2025年。[11] Matei Zaharia、Mosharaf Chowdhury、Tathagata Das、Ankur Dave、Justin Ma、Murphy McCauly、Michael J Franklin、Scott Shenker 和 Ion Stoica。弹性分布式数据集：用于 {容错} 的抽象，支持 { } *9th USENIX symposium on networked systems design and implementation* 内存中集 *(NSDI 12)* 15 28, 2012 群计算。载于，第–页。[12] Gyeong-In Yu、Joo Seong Jeong、Geon-Woo Kim、Soojeong Kim 和 Byung-Gon Chun。Orca：用于 { } *16th USENIX Symposium on Operating Systems* 基于 *Design and Implementation (OSDI 22)* 521 538, 2022 Transformer 的生成模型的分布式服务系统。载于，第–页。[13] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph Gonzalez、Hao Zhang 和 Ion Stoica。通过 PagedAttention 高效管理大语言模型服务中的内存。载于*Proceedings of the 29th symposium on operating systems principles*，第611–626页，2023年。

[14] Avrilia Floratou、Ashvin Agrawal、Bill Graham、Sriram Rao 和 Karthik Ramasamy。Dhalion：Heron 中的自调节流处理。 *Proc. VLDB Endow.*，10(12):1825–1836，2017年8月。[15] Vasiliki Kalavri、John Liagouris、Moritz Hoffmann、Desislava Dimitrova、Matthew Forshaw 和 Timothy Roscoe。你只需三步：为分布式流式数据流快速、准确地自动做出扩缩容决策。载于*13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*，第783–798页，2018年。[16] Jinqing Lian、Xinyi Zhang、Yingxia Shao、Zenglin Pu、Qingfeng Xiang、Yawen Li 和 Bin Cui。ContTune：通过保守贝叶斯优化持续调优分布式流数据处理系统。 *Proceedings of the VLDB Endowment* 16 13 4282 4295, 2023 ，():–。

[17] Bu˘gra Gedik、Scott Schneider、Martin Hirzel 和 Kun-Lung Wu。数据流处理的弹性伸缩。*IEEE Transactions on Parallel and Distributed Systems*，25(6):1447–1463，2014年。[18] Matthias Seeger。用于机器学习的高斯过程。 *International journal of neural systems* 14 02 69 106, ，():–。2004

[19] Jasper Snoek、Hugo Larochelle 和 Ryan P. Adams。机器学习算法的实用贝叶斯优化。载于 *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* 12, 2951 2959, 2012 ，NIPS’ 页–，美国纽约州 Red Hook，。Curran Associates Inc.[20] Lorenz Fischer、Shen Gao 和 Abraham Bernstein。机器调优机器：使用贝叶斯优化配置分布式流处理器。载于 *2015 IEEE International Conference on Cluster Computing*，第22–31页，2015年。[21] Michael A. Gelbart、Jasper Snoek 和 Ryan P. Adams。带有未知约束的贝叶斯优化。载于 *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI)*，2014年。[22] Daoyuan Chen、Yilun Huang、Zhijian Ma、Hesen Chen、Xuchen Pan、Ce Ge、Dawei Gao、Yuexiang Xie、Zhaoyang Liu、Jinyang Gao、Yaliang Li、Bolin Ding 和周靖人。Data-Juicer：面向大语言模型的一站式数据处理系统。载于 *SIGMOD*，2024年。[23] Philipp Moritz、Robert Nishihara、Stephanie Wang、Alexey Tumanov、Richard Liaw、Eric Liang、Melih Elibol、Zongheng Yang、William Paul、Michael I. Jordan 和 Ion Stoica。Ray：面向新兴人工智能应用的分布式框架。载于 *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*，OSDI’18，第561–577页，美国，2018年。USENIX协会。[24] Ke Cheng、Zhi Wang、Wen Hu、Tiannuo Yang、Jianguo Li 和 Sheng Zhang。SCOOT：面向大语言模型推理引擎的SLO性能调优。载于 *Proceedings of the ACM on Web Conference 2025*，WWW ’25，第829–839页，纽约，纽约州，美国，2025年。美国计算机协会。[25] James B. McQueen。多元观测的分类与分析方法。载于 *Proc. of 5th* ，第–页。*Berkeley Symposium on Math. Stat. and Prob.* 281 297, 1967

[26] Martin Ester、Hans-Peter Kriegel、J¨org Sander 和 Xiaowei Xu。一种在含噪大型空间数据库中发现聚类的基于密度的算法。载于*Discovery and Data Mining* 96, 226 231 1996 *Proceedings of the Second International Conference on Knowledge* ，KDD’ 页–。AAAI出版社，。[27] James Bergstra 和 Yoshua Bengio。用于超参数优化的随机搜索。 *Journal of machine learning research* 13 2 2012 ，(), 。

[28] Yanif Ahmad、Bradley Berg、U˘gur Cetintemel、Mark Humphrey、Jeong-Hyon Hwang、Anjali Jhingran、Anurag Maskey、Olga Papaemmanouil、Alexander Rasin、Nesime Tatbul 等。Borealis 流处理引擎中的分布式操作。载于*data* 882 884, 2005 *Proceedings of the 2005 ACM SIGMOD international conference on Management of* ，第–页。[29] Raul Castro Fernandez、Matteo Migliavacca、Evangelia Kalyvianaki 和 Peter Pietzuch。通过算子状态管理，在流处理中整合横向扩展与容错。载于 *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*，SIGMOD ’13，第725–736页，纽约，纽约州，美国，2013年。美国计算机协会。[30] Vincenzo Gulisano、Ricardo Jimenez-Peris、Marta Patiño-Martínez、Claudio Soriente 和 Patrick Valduriez。StreamCloud：一种弹性且可扩展的数据流系统。 *IEEE Trans. Parallel Distrib. Syst.*，23(12):2351–2365，2012年12月。

[31] Thomas Heinze、Zbigniew Jerzak、Gregor Hackenbroich 和 Christof Fetzer。面向分布式数据流处理系统的延迟感知弹性扩缩。载于 *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems* 14, 13 22, 2014 , DEBS '

页 – 纽约州纽约，美国，。美国计算机协会。[32] Le Xu、Boyang Peng 和 Indranil Gupta。Stela：使流处理系统能够按需缩容和扩容。载于 *2016 IEEE International Conference on Cloud Engineering (IC2E)*，第 22–31 页，2016年。[33] Yuan Mei、Luwei Cheng、Vanish Talwar、Michael Y. Levin、Gabriela Jacques-Silva、Nikhil Simha、Anirban Banerjee、Brian Smith、Tim Williamson、Serhat Yilmaz、Weitao Chen 和 Guoqiang Jerry Chen。Turbine：Facebook 面向流处理的服务管理平台。载于 *2020 IEEE 36th International Conference on Data Engineering (ICDE)* 1591 1602, 2020 ，第 – 页。

[34] Yang Liu、Huanle Xu 和 Wing Cheong Lau。具有遗憾保证的弹性流处理在线资源优化。载于 *Proceedings of the 51st International Conference on Parallel Processing*，ICPP '22，纽约州纽约，美国，2023年。美国计算机协会。[35] Dana Van Aken、Andrew Pavlo、Geoffrey J Gordon 和 Bohan Zhang。通过大规模机器学习自动调优数据库管理系统。载于 *Proceedings of the 2017 ACM international conference on management of data* 1009 1024, 2017 ，第 – 页。

[36] Omid Alipourfard、Hongqiang Harry Liu、Jianshu Chen、Shivaram Venkataraman、Minlan Yu 和 Ming Zhang。{ } *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* 469 482, 2017

CherryPick：自适应地发掘大数据分析的最佳云配置。载于 ，第 – 页。[37] Silvery Fu、Saurabh Gupta、Radhika Mittal 和 Sylvia Ratnasamy。关于使用（机器学习）进行黑盒系统性能预测。载于 *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)* 763 784, 2021 ，第 – 页。

[38] Zewen Yang、Dongfa Zhang、Xiaobing Dai、Fengyi Yu、Chi Zhang、Bingkun Huang、Hamid Sadeghian 和 Sami Haddadin。用于在线学习与控制的流式生成高斯过程专家：扩展版。 *arXiv preprint arXiv:2508.03679* 2025 ，。[39]

Mohammad Abu-Shaira、Ajita Rattani 和 Weishi Shi。DAO-GP：漂移感知在线非线性回归高斯过程。 *arXiv preprint arXiv:2512.08879*，2025年。[40] Xinyi Zhang、Hong Wu、Zhuo Chang、Shuowei Jin、Jian Tan、Feifei Li、Tieying Zhang 和 Bin Cui。ResTune：借助元学习增强的云数据库资源异向调优。载于 *Proceedings of the 2021 international conference on management of data* 2102 2114, 2021 ，第 – 页。

[41] Xinyi Zhang、Hong Wu、Yang Li、Jian Tan、Feifei Li 和 Bin Cui。迈向云数据库的动态安全配置调优。载于 *Proceedings of the 2022 International Conference on Management of Data*，第 631–645 页，2022年。[42] Konstantinos Kanellis、Cong Ding、Brian Kroth、Andreas M¨uller、Carlo Curino 和 Shivaram Venkataraman。LlamaTune：样本高效的数据库管理系统配置调优。 *arXiv preprint arXiv:2203.05128*，2022年。[43] Guoliang Li、Xuanhe Zhou、Shifu Li 和 Bo Gao。QTune：一种采用深度强化学习的查询感知数据库调优系统。 *Proceedings of the VLDB Endowment*，12(12):2118–2130，2019年。