MergePipe：面向可扩展大语言模型合并的预算感知参数管理系统

Yuanyi Wang 香港理工大学
yuan-yi.wang@connect.polyu.hk

Yanggan Gu 香港理工大学
yanggangu@outlook.com

Zihao Wang InfiX.ai，香港
wangzihao2020@lzu.edu.cn

Kunxi Li 浙江大学
kunxili@zju.edu.cn

Yifan Yang
香港理工大学
yi-fan.yang@connect.polyu.hk

Zhaoyi Yan InfiX.ai，香港
yanzhaoyi@outlook.com

Congkai Xie InfiX.ai，香港
xiecongkai@infix-ai.com

Jianmin Wu
香港理工大学
jianmin.wu@polyu.edu.hk

Hongxia Yang
香港理工大学
hongxia.yang1@gmail.com

arXiv:2602.13273v1 [cs.DB] 5 Feb 2026

摘要

大语言模型（LLM）合并已成为现代大语言模型开发流程中的一项关键技术，使得无需重新训练即可整合多个特定任务或领域的专家模型。然而，随着专家数量不断增加，现有的合并实现将模型参数视为无结构文件，并以无状态的一次性方式执行合并，导致磁盘输入/输出过多、参数扫描冗余且可扩展性较差。

本文介绍了**MergePipe**，一种面向可扩展大语言模型合并的参数管理系统。MergePipe 是首个将大语言模型合并视为数据管理与执行问题的系统，并针对模型参数、合并计划和执行沿袭引入了目录驱动抽象。MergePipe 的核心是成本感知规划器，它会显式建模专家参数输入/输出，并强制执行用户指定的输入/输出预算；随后，流式执行引擎会在事务保证下物化合并模型。我们的关键洞见是：虽然基础模型读取和输出写入不可避免，但专家参数读取占据合并成本的主要部分，因此是首要优化目标。通过在规划和执行的整个过程中使专家访问具备预算感知能力，MergePipe 缓解了朴素流水线的 $O(K)$ 输入/输出增长，并实现了可预测的扩展行为。实验表明，与最先进的大语言模型合并流水线相比，MergePipe 可将总 I/O 降低最多一个数量级，并实现最高 11× 的端到端加速（墙钟时间最多缩短 90%）。

1 引言

大语言模型（LLM）的开发日益从训练单一、庞大且稠密的模型，转向组合来自多个专家检查点或分支的模型 [22]。现有的大语言模型通常以检查点系列的形式维护，由一个通用基座模型和众多领域专用专家（例如数学 [25, 38], 代码 [11, 14], 法律 [7, 41],或生物医学领域的专家 [35]）组成，这些模型的训练规模达到数十亿至数万亿个词元 [1, 37]。当代开发流程越来越多地依赖**模型合并**来组合此类专家检查点，而不是从头重新开始训练，从而实现多技能整合、领域适配 [27, 31], 以及快速迭代，同时只需适量的额外计算资源 [3, 21]。与集成不同，集成会保留多个模型并在线汇总预测，因此会增加延迟和内存开销 [13], ；模型合并则直接作用于参数，生成单个可部署模型。因此，大语言模型合并已广泛应用于近期的大规模模型中，包括 DeepSeek-V3 [16] 和 Qwen3 [37],，合并被用于扩展训练、整合专门能力，或在开发过程中整合中间模型。

尽管其重要性日益凸显，现有的大语言模型合并流水线在很大程度上仍以与系统无关的方式实现。大多数现有实现将模型参数视为无结构的平面文件，并将合并作为无状态的一次性过程来执行 [20, 39]。每次合并调用都会独立加载专家参数、应用合并逻辑并生成输出，不会在不同合并之间复用，也没有显式的沿袭跟踪或任何成本感知计划的概念。因此，即使是在将同一基础模型与不断增加的专家集合进行合并时，系统仍会反复完整扫描专家参数。在现代大语言模型流水线中，检查点可能包含数百亿至数千亿个参数，这种设计使合并的输入/输出成本随专家数量线性增长，很快便令大规模合并变得昂贵且不切实际。重要的是，这些低效并非源于合并算法本身，而是因为缺少一种系统级抽象，将参数、合并计划和执行状态视为一等实体。图1和图2从流水线层面展示了这种差异，凸显了朴素合并在多个合并迭代中反复扫描专家参数的现象。具体而言，这要求围绕参数提供数据库风格的原语：针对参数块建立持久化目录，制定具有明确成本语义的声明式合并计划，并通过带有沿袭信息的事务性快照确保可复现性。

为理解这类执行模型为何难以扩展，我们考察大语言模型合并的底层成本结构。一次合并的端到端成本可分解为三个部分：（i）读取基础模型参数；（ii）生成合并后的输出检查点；（iii）读取专家参数。前两个部分是模型合并语义所固有的，因为任何合并操作都必须访问基础模型并生成新的检查点。相比之下，专家参数访问呈现出根本不同的行为。在朴素执行中，每次调用合并操作都会独立访问专家检查点，导致专家 I/O 成本

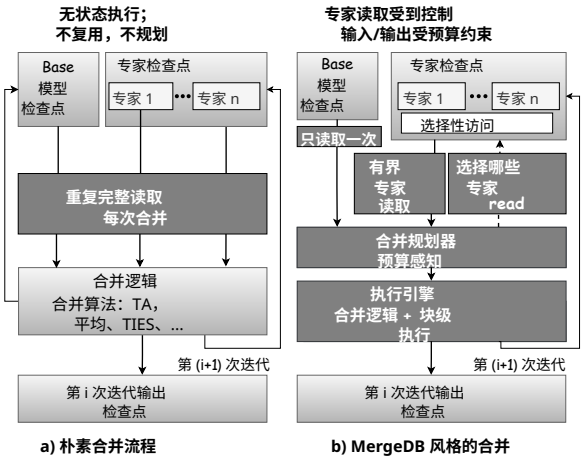


图 1：朴素方法的大语言模型合并与 MergePipe。朴素流水线以无状态、一次性的方式进行合并，并反复扫描专家检查点（$O(K)$ 专家 I/O）；而 MergePipe 引入了规划、复用和预算约束下的专家读取。

随专家数量 K 线性增长。这种 $O(K)$ 缩放很快就会主导总体合并成本，并占大规模合并工作负载中数据移动的大部分。这种分解揭示了系统设计中的一个关键不对称性：基础读取和输出写入是语义上的必需操作，而占主导地位的专家读取成本则是无状态流水线带来的执行层面产物，可通过规划与复用加以控制。换言之，专家成本是唯一一个其增长不由合并语义所决定的成本项 K，因此也是首要的优化目标。因此，相较于优化合并计算，控制专家参数访问才是实现可扩展大语言模型合并的主要策略。

然而，仅仅认识到专家参数访问是主要成本，并不足以在实践中实现可扩展的合并。不同合并算法和工作负载的专家访问模式差异显著。一些合并算法会对所有参数进行统一操作，例如模型汤 [34] 和任务算术（TA）[15], ; 另一些则会根据冲突检测或相似度阈值有选择地访问专家参数，例如 TIES [36] 和 DARE [40]。因此，减少专家 I/O 所能带来的收益因合并策略而异，无法通过固定启发式方法或临时缓存机制来充分体现。因此，有效的优化需要一种严谨的方法，针对不同的合并计划对专家访问成本进行建模、预测和推理。

此外，在合并过程中引入复用、一致性和沿袭跟踪不可避免地会产生系统级开销。尽管在小规模合并场景中，这类开销可能超过优化带来的收益，但随着专家检查点数量或大小的增长，这些开销便可得到摊销。这种权衡意味着，不存在一种适用于所有场景的最优执行策略。因此，大语言模型合并需要一种能够明确呈现并控制这些权衡的系统，使用户能够根据工作负载特征，在专家访问和执行开销之间进行平衡。

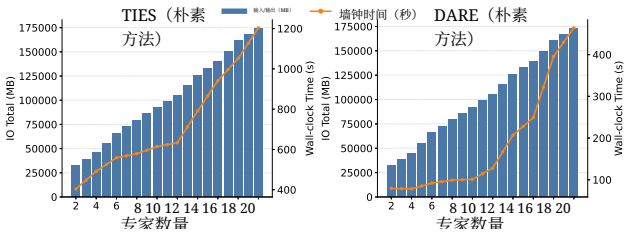


图2：朴素合并随着专家数量增加而扩展性不佳。对于3B模型，TIES 和 DARE 的总 I/O（柱状图）均随着 K 增加而近似线性增长，墙钟时间也随之上升。

为应对这些挑战，我们提出了**MergePipe**，这是一种面向可扩展且可复现的大语言模型合并的参数管理系统。MergePipe 并未引入新的合并算法，而是将模型合并重新定义为一个数据管理问题，并为模型参数、合并计划和执行状态提供数据库支持的抽象。在 MergePipe 中，合并计划被视为一等对象，明确记录访问哪些参数、如何组合这些参数，以及需要满足哪些资源约束。这种设计将参数存储、合并规划和执行解耦，从而实现跨合并的规范化复用，并对专家访问进行明确控制。具体而言，MergePipe 引入了一个成本感知规划框架，将专家参数输入/输出建模为一种明确设定预算的资源。给定用户指定的输入/输出预算，规划器会推导出有选择地安排专家访问、同时保留合并语义的执行计划。执行引擎负责落实这些计划，并提供事务性写入和细粒度沿袭跟踪，从而确保正确性、可复现性以及跨合并迭代的增量复用。

总而言之，本研究揭示了三个关键洞见：

- **I1：专家读取主导扩展性，而非合并计算**。朴素流水线会在每次调用时反复扫描专家检查点，导致专家 I/O 随专家数量 K 近似线性增长（图1、图2）。
- **I2：专家 I/O 是大语言模型合并中首要的可控成本因素**。基础模型读取和输出写入是合并流水线中固有的开销，而可扩展合并的关键在于通过复用和预算感知访问来减少专家读取（图4）。
- **I3：预算感知规划实现了可预测且与算子无关的控制**。通过将专家参数访问建模为一种显式资源，MergePipe 可在适用于各种合并算子的同时，确保遵守用户指定的 I/O预算（表4）。

我们的贡献如下：(1) MergePipe，这是首个面向迭代式大语言模型合并的参数管理系统，它基于目录构建数据模型，涵盖参数块、合并计划和执行数据沿袭；(2) 一种预算感知规划框架，将成本模型与可执行计划绑定，通过明确设定专家读取预算，消除朴素流水线的 $O(K)$ 专家扫描行为；(3) 一种执行引擎，支持事务性快照和细粒度数据沿袭，能够在规划预算范围内，以可复现、可审计的方式物化合并检查点。

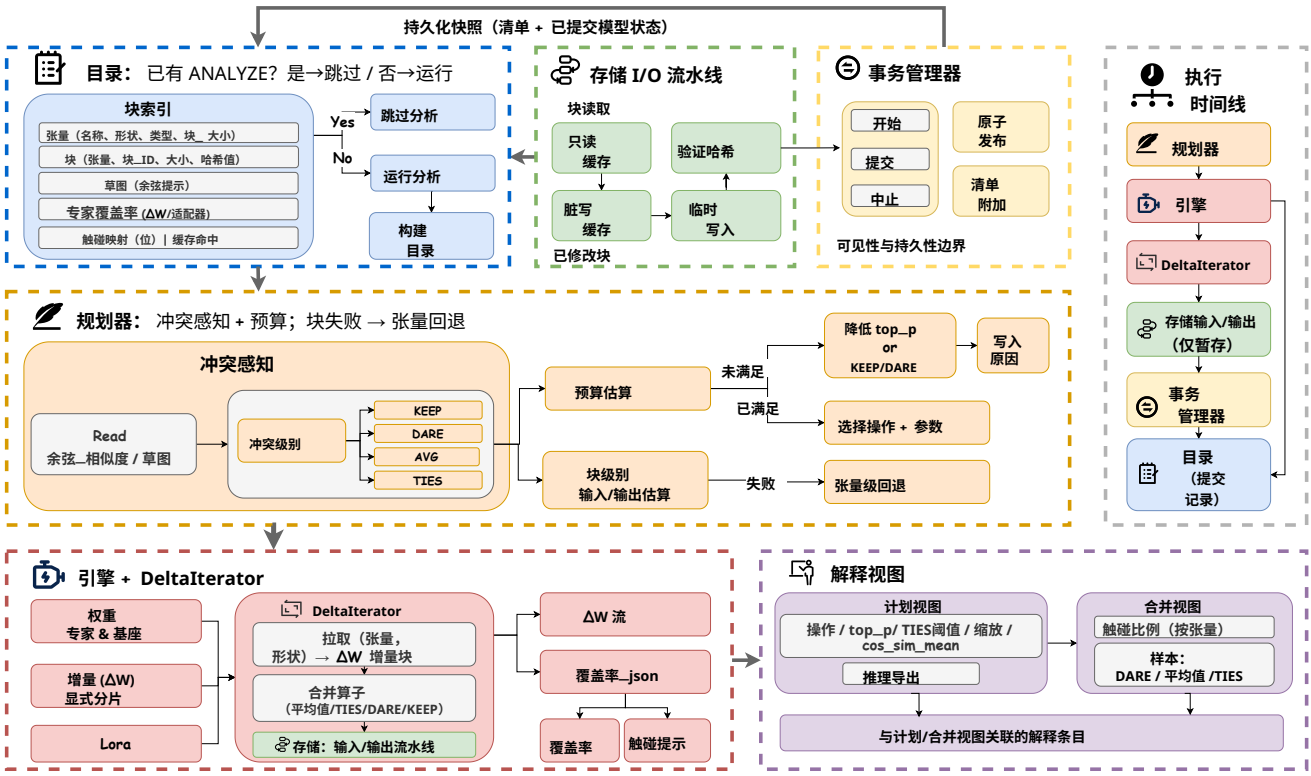


图3：MergePipe 系统概览。MergePipe 将参数存储、预算感知规划与流式执行解耦，以实现可扩展大语言模型合并。持久化目录支持块级索引/复用；成本与冲突感知规划器在显式 I/O预算下选择专家块；执行引擎通过 DeltaIterator 强制执行该计划，并配合原子发布以及数据沿袭/可解释性记录。

2 系统概览

为应对大规模大语言模型合并中的可扩展性和可复现性挑战（第 1 节），我们提出了**MergePipe**，一种将合并视为数据管理工作负载的参数管理系统。MergePipe 不针对特定合并算法进行优化，而是通过解耦参数存储、合并规划和执行，着眼于底层执行模型。图 3 概述了该系统：(i) 持久化目录和存储输入/输出流水线；(ii) 成本感知且冲突感知的规划器；(iii) 流式执行引擎；以及 (iv) 用于数据沿袭的事务和可解释性组件。这些组件共同支持有预算约束的专家访问、跨迭代复用以及可审计的合并。

2.1 设计目标与范围

MergePipe 面向迭代合并：在受限的输入/输出条件下，将缓慢演进的基础模型反复与不断增长的专家集合进行合并。在第3节成本结构的指导下，我们采用：**G1：有界专家 I/O**。 专家读取主导合并成本，并在朴素流水线中按 $O(K)$ 增长；MergePipe 将专家访问视为预算化资源。

G2：执行前进行显式规划。MergePipe 将规划与执行分离。计划指定访问的参数和资源约束，从而支持成本估算、复用和预算感知回退。

G3：持久化执行。 MergePipe 持久记录决策、访问和数据沿袭，以支持可审计性、复用和重新计算。

范围：MergePipe 管理并执行合并，而不引入新的合并算法。它对算子无关，例如 TIES、DARE，并支持完整权重、参数增量以及基于适配器的专家。我们聚焦于磁盘主导型场景，其中参数输入/输出是瓶颈，而非训练时内存融合。

2.2 数据模型和目录

MergePipe 围绕一个持久化目录构建，用于将模型参数和合并相关元数据作为一等数据进行管理。该目录提供：(i) 用于成本估算和复用的块粒度视图；(ii) 配有清单的不可变快照，以支持可复现的执行。**块：** 模型检查点 M 由命名张量组成。每个张量 T通过确定性函数 Partition($T; s$) 划分为定长块 $\{b\}$，其中 s为块大小。块标识符是一个稳定的标识符 模型_编号，张量_编号，块_索引，可唯一定位存储中的物理块。

表1：MergePipe 用于区块级规划、复用和可复现快照的核心目录记录。

记录	关键字段（主键以下划线标出）
区块元数据	模型_标识、张量_标识、区块_索引；字节数、形状、数据类型，哈希值、草图、布局
触碰映射	会话标识符、张量_标识；已触碰的_块（位图 / 范围）
覆盖率	会话标识符、张量_标识、块_索引；专家_集合_摘要（或列表）
PLAN	计划_标识；基座_标识、专家_标识， op， 预算_B，已选-中的_块_摘要， $\widehat{C}_{\text{expert}}$
清单	会话标识符；计划_ID、基座_ID、专家_ID、操作、预算_B，（已实现-化） C_{expert}，输出_根目录，创建_时间

快照与清单： 一次已提交的合并会生成一个由 *sid* 标识的快照以及一个清单 man(*sid*)。man(*sid*) 是一种不可变映射，将每个张量区块键 张量_编号，块_索引映射到一个存储对象引用（以及执行元数据，例如输入、算子和计划摘要）。快照是复用、检查和增量重计算的基本单位。

不变量（不可变性与原子可见性）。已提交快照不可变：一旦清单（*sid*）发布，其区块映射和元数据便永不更改。每次合并执行要么原子地发布一个包含完整清单的新快照 *sid′*，要么什么也不发布（即不会出现部分可见的情况）。

核心目录记录。 为支持区块级成本估算、复用检测和可审计血缘，MergePipe 会存储一小组结构化记录。表1总结了核心关系（逻辑模式；物理后端可以是键值存储或嵌入式数据库）。

目录如何支持规划与复用。 BlockMeta 提供区块级成本核算和轻量级冲突信号所需的最少统计信息。Plan 记录在预算约束下选定的专家块，并可在迭代合并中复用。TouchMap 和 Coverage 汇总了实际物化的内容以及做出贡献的专家，从而支持可解释性和增量重计算。Manifest 将计划绑定到不可变快照边界，使合并可复现且可审计。

2.3 MergePipe 合并规划与执行工作流

MergePipe 按照分阶段工作流处理不可变快照（第2.2节）。分析和规划仅操作元数据，而执行则以原子方式发布新快照（图3）。

分析。 MergePipe 会检查目录中是否已有分析结果；若未命中，则计算并持久化区块级元数据，例如形状、哈希值、草图和覆盖率，以便复用。

规划。 给定一个合并请求和专家 I/O 预算，规划器会根据目录元数据估算专家访问成本，并在可行时生成块粒度的预算可行计划（并支持张量级回退）。

执行与提交。 引擎通过 DeltaIterator 流式传输块，逐步应用合并算子，并仅物化选定块，从而强制执行计划。写入会暂存，并通过事务管理器原子发布；同时记录触及映射和覆盖摘要，以提升可解释性并实现复用。通过将分析、规划和执行分离，MergePipe 实现了可预测的资源使用、增量复用和可复现的合并，同时不对合并算子施加限制。

3 模型合并的成本模型

本节介绍一种用于大规模大语言模型合并的成本模型。该模型的目标并非预测精确的执行时间，而是捕捉决定受限 I/O 资源条件下可扩展性的主要成本因素。该模型提供了一种有原则的抽象，为 MergePipe 的规划器和执行设计奠定基础。

3.1 概述与假设

我们考虑将一个基座模型与一组专家模型合并的问题。设

$$\mathcal{M} = \{M_0, M_1, \ldots, M_K\}$$

表示参与合并的模型，其中 M_0 是基座模型， M_1、.....、 M_K 是专家模型。每个模型由一组张量组成：

$$M_i = \{T_{i,1}, T_{i,2}, \ldots, T_{i,n_i}\}.$$

每个张量都持久存储在磁盘上；我们用 size(T) 表示其物理大小。

我们重点关注磁盘 I/O 成本，因为它是大规模合并中的主要瓶颈。这一假设与实证观察一致：合并工作负载需要读取和写入数十到数百 GB 的参数，而计算的开销相对较轻。MergePipe 的一个关键区别在于，模型合并语义固有的成本与可通过规划决策控制的成本并不相同。只有后者才是优化的对象。

3.2 成本分解

我们将合并操作的总成本分解如下：

$$C_{\text{merge}} = C_{\text{base}} + C_{\text{expert}} + C_{\text{out}} + C_{\text{meta}}.$$

各组成部分定义如下。

基础模型读取成本。 要构建合并模型，必须完整读取基础模型：

$$C_{\text{base}} = \sum_{T \in M_0} \text{size}(T).$$

这项成本在不同合并策略之间保持不变，且与专家数量无关。

输出物化成本。合并模型必须完全物化并写入存储：

$$C_{\text{out}} = \sum_{T \in M_{\text{merged}}} \text{size}(T).$$

合并模型保留了基础模型的张量结构。**元数据成本**。 元数据操作包括血缘记录、哈希计算和事务提交：

$$C_{\text{meta}} = C_{\text{lineage}} + C_{\text{hash}} + C_{\text{commit}}.$$

这些因素有界，且与合并策略的相关性较弱 .

专家读取成本。 占主导地位且可控的部分是专家读取成本：

$$C_{\text{expert}} = \sum_{i=1}^{K} \sum_{T \in \mathcal{S}_i} \text{size}(T),$$

其中，$\mathrm{S}_i \subseteq M_i$ 表示从专家M_i读取的张量子集。在朴素合并流水线中，会读取所有专家张量：

$$C_{\text{expert}}^{\text{naive}} = \sum_{i=1}^{K} \sum_{T \in M_i} \text{size}(T),$$

其随专家数量 K线性增长。相比之下，MergePipe旨在通过有选择地访问专家参数来降低C_{expert}。

3.3 块级成本估算

MergePipe通过以块粒度进行操作，细化了专家成本估算。每个张量 T都被划分为一组定长块 $T = \{b_1, \ldots\ldots, b_m\}$。专家读取成本可表示为：

$$C_{\text{expert}} = \sum_{i=1}^{K} \sum_{b \in \mathcal{B}_i} \text{size}(b),$$

其中，B_i 表示从专家 M_i中选定的块。

张量级估算是该公式的一种特殊情况，其中会选择张量的所有块。块级估算能够对专家访问进行更精细的控制，并为 MergePipe 的规划和回退机制奠定基础。

3.4 预算约束下的规划目标

MergePipe 对专家读取引入了明确的 I/O 预算约束：

$$C_{\text{expert}} \le B,$$

其中， B 是用户指定的预算（例如以 MB 为单位）。

该预算仅适用于专家读取。基础模型读取、输出物化和元数据操作均视为固定开销，不受优化影响。在此约束下，规划器旨在选择满足预算要求的专家块 $\{\mathrm{B}_i\}$，同时遵循合并专用启发式方法：

$$\min \; C_{\text{expert}} \quad \text{s.t.} \quad C_{\text{expert}} \le B.$$

该优化问题并非通过精确求解得出。相反，成本模型提供了一种结构化抽象，用于指导第 4 节所述的预算感知规划启发式方法。通过区分固定成本和可控成本，该模型使扩展行为能够随着专家数量的增加而保持可预测：朴素流水线的专家 I/O 增长为$O(K)$，而 MergePipe 则将 I/O 限制在与 K 无关的范围内。

4 合并规划

本节介绍 MergePipe 规划器的设计，该规划器基于第 3 节中的成本模型生成预算感知的合并计划。规划器的目标是在保留指定合并算子语义的同时，控制对专家参数的访问。规划器并不优化合并计算，而是明确考虑专家 I/O 成本，并强制执行用户指定预算。

4.1 规划器接口与范围

规划器的输入包括：(i) 一个合并请求，指定一个基座模型、一组专家模型以及 AVG、TIES、DARE 等合并算子；(ii) 用户指定的专家 I/O 预算 B；以及 (iii) 先前 ANALYZE 阶段生成的目录元数据（第2.2节）。

规划器输出一个合并计划，以声明方式指定：(a) 访问哪些专家参数（或块），(b) 应用哪个合并算子及其参数，以及 (c) 预期的专家读取成本。该计划是一种一等对象，可以由执行引擎独立检查、复用和执行。

规划器的范围。 规划器不会修改合并公式，也不会引入新的合并算法。相反，它会在受限资源下确定访问哪些参数。这种分离确保 MergePipe 不依赖于合并算子，同时提供可预测的系统级行为。我们将合并计划形式化为一份明确的选择与执行契约。

定义 *4.1*（合并计划）。合并计划是一个元组

$$\pi = \big(\text{op}, \theta, \{\mathcal{B}_i\}_{i=1}^{K}, \text{order}\big),$$

其中，op 是合并算子， θ 表示算子参数，B_i是从专家M_i中选取并访问的参数块集合，order 是执行引擎使用的确定性张量或块遍历顺序。

定义 *4.2*（可行性）。设 $\widehat{C}_{\text{expert}}(\pi) = \sum_{i=1}^{K} \sum_{b \in \mathcal{B}_i} \text{size}(b)$ 为块级估算下计划的专家读取成本。若计划 π 满足专家 I/O 预算 B，则该计划符合预算，当且仅当

$$\widehat{C}_{\text{expert}}(\pi) \le B.$$

4.2 将成本绑定到模型

规划器通过将每个候选计划 π 绑定到估算的专家读取成本 $\widehat{C}_{\text{expert}}(\pi)$，来实例化第 3 节中的成本模型。$\widehat{\ }$只有 $C_{\text{expert}}(\pi)$ 会参与可行性检查，而 C_{base}、 C_{out}和 C_{meta} 则视为固定开销。只有满足定义 4.2 中预算可行性条件的计划才会被接受。

4.3 冲突感知信号

为了在预算约束下优先处理专家参数，规划器利用源自目录元数据的冲突感知信号。这些信号包括轻量级草图（例如余弦相似度提示）以及在ANALYZE期间计算的覆盖率统计。直观而言，这些信号用于估算专家参数与基础模型的差异程度，或其与其他专家的重叠程度。规划器仅使用这些信号进行排序和选择。它们不会改变合并算子的语义。例如，在 AVG 合并中，选中的参数仍会取平均值；在 TIES 或 DARE 中，则仍应用相同的阈值处理或掩码规则。资源有限时，冲突感知信号只影响访问哪些参数。

4.4 预算感知计划生成

MergePipe 使用贪心过程，在明确的预算 B 下逐步选择专家块来生成计划。算法1对此过程进行了形式化：规划器 (i) 利用目录信号（例如草图和覆盖率）为每个候选块分配优先级分数 $s(i, b)$，(ii) 按分数降序遍历候选项，(iii) 添加

算法1： PlanGen：贪心预算感知计划。

输入： 专家 $\{M_i\}_{i=1}^{K}$；目录 C；算子 操作；预算 B
输出： 一个预算可行计划 π
为每个候选专家计算优先级分数 $s(i,\ b)$
块 b 使用目录草图/覆盖率
按降序排列候选项 $s(i,\ b)$
为所有 i 初始化 $\mathrm{B}_i \leftarrow \emptyset$，并计算成本 $\leftarrow 0$
遍历 候选项 (i, b)，按排序顺序 **执行**
  如果 成本 + 大小$(b) \le B$ **则**
    $\mathrm{B}_i \leftarrow \mathrm{B}_i \cup \{b\}$；成本 $\leftarrow$ 成本 +大小(b)
  end
  可选地调整 θ 以用于操作 并继续，否则跳过
end
返回 $\pi =$(操作 θ，$\{\mathrm{B}_i\}$，顺序)

当且仅当增量成本能够保证可行性时，才将块加入计划，即成本 +size$(b) \le B$。如果下一个候选项会超出预算，规划器则跳过该块，或在算子支持的情况下，对算子参数进行有限调整（例如降低 TIES/DARE 的阈值 θ）；所有此类决策都会记录在计划元数据中，以确保可复现性。由此生成的计划 π 从构造上即满足预算约束，并在 B 范围内最大限度地利用高优先级专家信息。

4.5 回退、鲁棒性与复杂度

当区块级元数据缺失或不可靠时，规划器会回退到更粗粒度的张量级选择，即选择整个张量（等价于选择所选张量中的所有块），并使用第 3 节中的张量级公式估算 $\widehat{C}_{\text{expert}}(\pi)$。为确保可复现性，回退事件及其原因会记录在计划元数据中。令 N_b 表示规划期间考虑的参数块数量。由于优先级排序和预算检查，规划器的运行时间为 $O(N_b \log N_b)$。在实践中，与执行阶段的输入/输出相比，规划开销可以忽略不计；通过在迭代式合并工作流中复用计划，还可将其摊销。

5 执行引擎

MergePipe 的执行引擎会强制执行规划器生成的合并计划 π，并在明确的专家 I/O 预算下生成合并后的快照。算法 2 定义了执行语义。

5.1 执行语义与预算健全性

定义（实际读取的专家）。令 $\mathrm{Sel}_\pi(t,\ b) \subseteq \{1,\ \ldots\ldots,\ K\}$ 为 π 为输出块（$t,\ b$）所选的专家。算法 2 实际读取的专家集合为

$$\mathcal{R}_{\text{expert}}(\pi) = \{(i,t,b) \mid i \in \mathrm{Sel}_\pi(t,b)\}, \tag{1}$$

$$C^{\text{run}}_{\text{expert}}(\pi) = \sum_{(i,t,b)\in\mathcal{R}_{\text{expert}}(\pi)} \text{size}(b). \tag{2}$$

输出快照是一个完整检查点：对于每个基础块（$t,\ b$），引擎读取 x_0 并写入输出块 x（算法 2，步骤 (1)）；只有专家块会通过 D.Pull(b) 进行选择性读取。

算法 2： ExecuteMerge：预算约束的流式执行与原子提交。

输入： 合并计划 π；基础模型 M_0；专家 $\{M_i\}_{i=1}^{K}$；
存储 S；目录 C；事务管理器 T
输出： 已提交快照 ID sid 和清单 man
事务与暂存。
T .开始()
$w \leftarrow$S.Open暂存写入器()
触及 $\leftarrow \emptyset$;覆盖率 $\leftarrow \emptyset$
(1) 根据计划流式处理选定块 π。
遍历 张量 $t \in \pi$.张量顺序() **执行**
  $D \leftarrow$ InitDeltaIterator($t,\ \pi,\ M_0,\ \{M_i\}$)
  对于每个 块 $b \in \pi$.待物化块(t) **执行**
    $\{\Delta_i\} \leftarrow D$.拉取($b$)　　// 专家块当且仅当已选择
    $x_0 \leftarrow$ S.读取基础块(M_0, t, b)
    $x \leftarrow$ 应用算子($x_0,\ \{\Delta_i\}$ π.操作($t,\ b$))
    w.写入块($t,\ b,\ x$)
    触摸$[t] \leftarrow$ 触摸$[t] \cup \{b\}$
    覆盖率$[t,b] \leftarrow D$.已使用专家()
  end
end
(2) 验证并原子发布快照。
S。验证哈希(w)
清单 $\leftarrow$ C.构建清单(π,触及,覆盖率)
$sid \leftarrow$ T.原子发布(w,清单)
C.提交记录(sid,清单)
T .提交()
返回 (sid,清单)

性质（预算可靠性）。在步骤 (1) 中，当且仅当 D.Pull(b)当且仅当 $(i,\ t,\ b) \in \mathcal{R}^{\text{expert}}(\pi)$ 时，引擎才会调用该操作，因此实际产生的专家输入/输出受计划和用户预算约束：

$$C^{\text{run}}_{\text{expert}}(\pi) \ \le\ \widehat{C}_{\text{expert}}(\pi) \ \le\ B,$$

精确到存储层所采用的核算粒度，例如缓冲 I/O 中按块大小取整。这一属性直接成立，因为 π 会明确枚举引擎获准访问的所有专家块，而且引擎不会执行超出 π所选范围的任何额外专家读取。

5.2 DeltaIterator 与算子应用

为支持异构专家（完整权重、增量、适配器），引擎使用统一的 DeltaIterator。对于张量 t，InitDeltaIterator 构造 D，使得 D.Pull(b) 恰好返回块 b所选专家的贡献$\{\Delta_i\}$（算法 2，步骤 (1)）。引擎通过以下方式计算每个输出块

$$x \leftarrow \textsc{ApplyOperator}(x_0, \{\Delta_i\}, \pi.\textsc{Op}(t,b)),$$

不修改算子语义；π仅控制访问哪些专家信息。

表 2：实验中评估的模型配置。所有模型均使用 HuggingFace 官方检查点。BF16 大小按每个参数 2 字节估算，反映合并期间参数输入/输出的大致规模。

模型	参数量	层数	隐藏维度	BF16 大小	专家数量
Qwen3-0.6B	0.6B	24	1024	~1.2 GB	20
Qwen3-1.7B	1.7B	28	2048	~3.4 GB	20
Qwen3-8B	8.0B	32	4096	~16.0 GB	20
Llama-3.2-3B	3.0B	26	3072	~6.0 GB	25
Llama-3.1-8B	8.0B	32	4096	~16.0 GB	20

5.3 原子发布与数据沿袭清单

所有写入都会先暂存并经过哈希值验证，然后才发布（算法 2，步骤 (2)）。随后，引擎会构建清单 man $\leftarrow$ C。BuildManifest(π、*touch*、覆盖率)，并通过 T。AtomicPublish(w、man) 原子发布新的快照 ID sid。

属性（原子可见性）。算法 2 保证一次合并要么提交一个新的快照（sid，man），要么不留下任何外部可见的部分状态。

6 实验

本节旨在通过实证验证第 1 节和第 3 节中提出的关键见解，并展示 MergePipe 如何通过明确控制专家参数访问，实现可预测且可扩展的性能。具体而言，我们的实验围绕以下见解展开：

- **I1. 专家 I/O 主导可扩展性：**朴素合并流水线会反复扫描专家检查点，导致专家 I/O 和端到端耗时几乎随专家数量线性增长。
- **I2. 专家 I/O 是主要的可控成本杠杆：**虽然基础模型读取和输出写入不可避免，但有选择地减少专家读取，是在不修改合并语义的前提下实现可扩展合并的关键。
- **I3. 预算感知规划可实现可预测的控制：**将专家 I/O 显式建模为预算资源，使我们的 MergePipe 能够适用于不同的合并算子和模型系列。
- **I4. 系统开销很小且可摊销：**规划、元数据管理和事务执行引入的额外成本，相较于端到端合并耗时微不足道，并且在实际规模下可以摊销。
- **I5. 块粒度适用于稳健的运行区间：**在实际可用范围内，MergePipe 的性能在不同块大小和执行设置下保持稳定，这表明其优势并不依赖脆弱的调优。

所有实验均未修改合并算法本身，并直接体现了第 3 节中介绍的成本分解和预算约束规划目标，尤其关注控制专家读取成本 C_{expert}。[1]

[1]我们的项目发布于 https://github.com/wyy-code/mergepipe。

6.1 实验设置

模型与专家。我们在 Llama 和 Qwen 系列的五个代表性开源大语言模型上评估 MergePipe：{*Llama-3.1-8B*、*Llama-3.2-3B*、*Qwen3-0.6B*、*Qwen3-1.7B*和*Qwen3-8B*}。这些模型的参数规模从 0.6B 到 8B 不等。所有实验均使用 HuggingFace 官方检查点。表 2 总结了模型的主要特征，包括参数量和 BF16 检查点的大致大小（按每个参数占 2 字节估算）。

对于每个基座模型，我们通过逐步增加专家检查点的数量来构建合并工作负载。除非另有说明，专家均由同一基座模型派生，并以完整检查点或参数增量的形式存储。

合并算子。我们使用两种广泛采用的稀疏合并算法 TIES 和 DARE 评估 MergePipe。这两种方法都会选择性地纳入专家参数，因此能够自然地受益于预算感知的专家访问。尽管 MergePipe 也支持 Average 合并，但我们的评估重点关注 TIES 和 DARE，因为它们更能反映现实中的大规模合并工作流程。

基线。我们将 MergePipe 与实现相同合并算子、但将模型参数视为非结构化文件的朴素合并流水线进行比较。每次合并调用都会独立扫描所有所需的专家参数，不进行复用或规划，也不执行预算约束，这反映了现有开源合并脚本的执行模型。

指标。我们报告 (i) **专家读取 I/O**(MB)、(ii)**端到端墙钟时间**和 (iii) 相对于朴素执行的**相对改进**。这些指标直接对应第 3 节中定义的成本组成部分，并重点关注占主导地位的专家读取成本 C_{expert}。

系统配置。所有实验均在一台仅使用 CPU 的服务器上进行，该服务器配备两颗 Intel Xeon Platinum 8358 处理器（共 64 个物理核心、128 个硬件线程）和固态硬盘存储。除非另有说明，否则禁用操作系统级文件缓存。不使用 GPU 加速，以确保合并性能受 I/O 限制，而非受计算限制。

可复现性。MergePipe 会持久化合并计划、执行清单和血缘元数据，从而支持确定性重新执行和事后检查。

6.2 I1.随专家数量缩放

我们研究随着专家集增大，MergePipe 的缩放表现如何：使用 TIES 作为合并算子，并将专家数量从 2 个逐步增加到 20 个（Llama-3.2-3B 则增加到 25 个）。图 4 汇总了五个模型系列中的专家 I/O、端到端墙钟时间和总 I/O。

专家 I/O 从 $O(K)$ 扩展到预算值。图 4(a，上)显示，朴素流水线的专家读取 I/O 会随着 K 增加而近乎线性增长，这与我们成本模型（第 3 节）中的 $O(K)$ 项 C_{expert} 相符。相比之下，MergePipe 通过强制执行专家读取预算 $C_{\text{expert}} \le B$，使专家 I/O 在几乎保持平稳的情况下适用于 K，并在最大 K 时大幅降低开销：例如，对于 Qwen3-8B 和 Llama-3.1-8B，在 K=20 时，朴素方法的专家 I/O 可达数百 GB，而 MergePipe 则保持在数十 GB 范围内（约降低一个数量级）。

墙钟时间与专家输入/输出趋势一致，仅有少数局部异常值。图 4(a)底部报告了实际运行时间。总体而言，墙钟时间与专家输入/输出趋势一致：随着 K 增大，朴素合并的速度持续下降，而

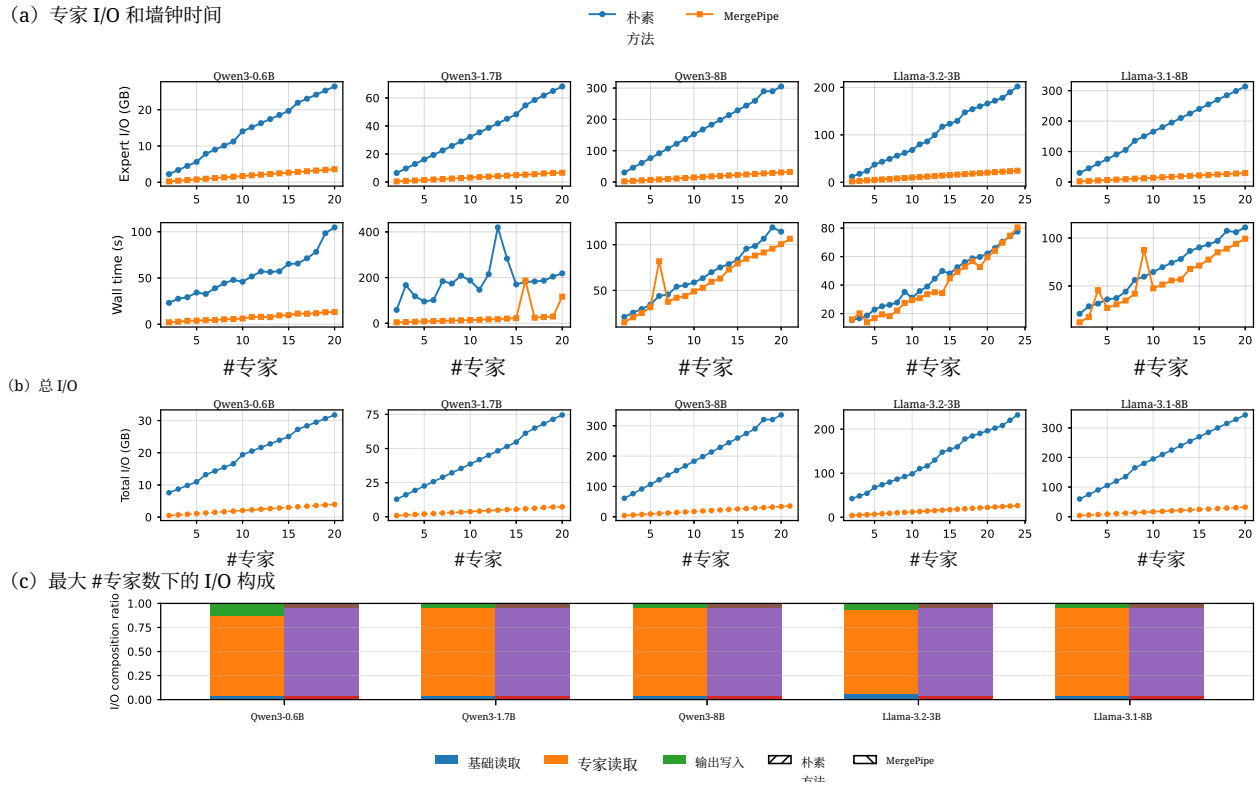


图 4：随专家数量扩展的变化。（a）专家读取 I/O（上）和端到端墙钟时间（下）。朴素合并会反复扫描专家检查点，导致随着专家数量增加，专家 I/O 和运行时间近似线性增长。MergePipe 在执行时强制实施明确的专家 I/O 预算，使专家读取保持在受控范围内，并显著减少墙钟时间。（b）总 I/O 表明，朴素流水线逐渐由专家读取主导，而 MergePipe 通过限制专家访问减少了总 I/O。（c）最大专家数量下的 I/O 构成表明，MergePipe 将主要成本从专家读取转移至不可避免的基础读取和输出写入。所有实验均在 CPU 上使用相同的合并算子（TIES）进行。

对于所有模型和较大的 K，MergePipe 仍显著更快（例如，在 Qwen3-1.7B 和 Qwen3-8B 上优势最大，因为朴素方法会反复扫描大型专家检查点）。我们观察到若干局部峰值，成本模型无法解释这些峰值（因为它们并未伴随专家输入/输出增加）：(i) *Qwen3-1.7B*（朴素方法）在 $K \approx 13$–14附近出现了明显的墙钟时间峰值；(ii) *Qwen3- 8B*（*MergePipe*）在 $K \approx 6$附近出现了孤立峰值；(iii) *Llama-3.1- 8B*（*MergePipe*）在 $K \approx 9$–10附近出现了峰值。这些异常值符合瞬时系统级干扰（CPU 调度与频率调节、后台磁盘争用以及 NUMA 效应）的特征，而非合并语义导致：重要的是，这些异常发生时专家输入/输出并未相应跃升，曲线也会立即恢复此前的趋势。即使在这些点上，对于大型模型，MergePipe 在相近的 K 下仍远快于朴素方法。

总 I/O 开始由不可避免的开销主导。 图 4(b) 展示了总 I/O。朴素方法的总 I/O 几乎随 K be- cause 专家读取的增加而线性增长，因为专家读取主导了 C_{merge}。使用 MergePipe 时，总 I/O 增长缓慢，并且越来越多地由固定开销 C_{base} 和 C_{out}主导。图 4(c) 清晰地展示了这一转变：在最大 K下，朴素方法的运行主要由专家读取主导，而 MergePipe 则大幅降低了专家读取占比，使基础读取和输出物化成为主要开销。

这些收益从何而来？ 为回应关于性能改进可能源于实现细节而非我们的执行模型的疑虑，我们进一步分解了 I/O 和开销。图 5 显示：（i）预算前的专家读取随着 K 在不同模型间近似线性扩展；（ii）改变预算主要影响专家读取，而基础读取和输出写入几乎保持不变；（iii）在实际预算范围内，规划+刷新+提交仅占墙钟时间的一小部分。综合来看，这些结果表明，MergePipe 的收益主要来自减少占主导地位的专家读取项，而非减少元数据或事务开销。**基线强化。** 为降低我们的收益源于基线工程不足的风险，我们还在相同的指标接口下报告了禁用预算的消融实验。表3显示，MergePipe 仍显著优于朴素全量读取；在相同 I/O 上限下，禁用规划器侧预算缩放会导致运行时间变长。

随着专家数量增加，朴素合并很快会因 $O(K)$ 次专家扫描而变得 I/O 受限。MergePipe 强制执行

表3：基线与禁用预算消融（Qwen3-0.6B，K=20，预算=8.57GB）。与朴素全量读取相比，MergePipe 的优势依然存在；在相同的 I/O 上限下，禁用规划器侧预算缩放会增加墙钟时间。

方法	专家读取（GB）	总 I/O（GB）	耗时（秒）
朴素方法（完整读取）	26.4	31.8	104.7
MergePipe（预算控制）	8.0	8.8	22.8
MergePipe（禁用预算）	8.0	8.8	34.5

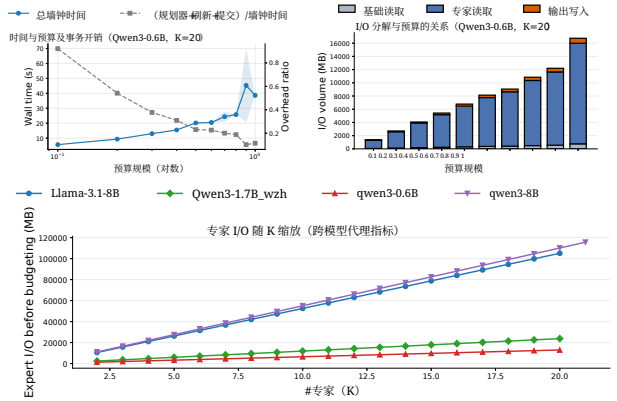


图5：（a）规划、刷新和提交仅占墙钟时间的一小部分。（b）预算主要减少专家读取。（c）预算前的专家读取 I/O 随模型数量 K 变化。

第 3 节中的预算化专家读取项，实现了可控的缩放，同时大幅降低了专家 I/O 和端到端时间；剩余开销则越来越受不可避免的基础读取和输出写入开销限制。在所有模型和专家规模下，相对趋势均保持一致，这表明观察到的收益源自执行模型，而非特定模型的影响。

6.3 I2. 预算感知的规划行为

接下来，我们研究在明确的 I/O预算约束下，MergePipe 的规划器如何运行。在固定基座模型、合并算子（TIES）和专家数量的情况下，我们将专家读取预算从完整专家参数大小的 10% 调整到 100%。图6总结了专家 I/O、端到端延迟和规划器访问行为。

专家 I/O 遵循施加的预算。 图6(a)展示了专家数量为 10 和 20 的工作负载在不同预算下的专家读取总量。实际专家 I/O 随预算单调增加，且始终低于理论预算上限（虚线），这证实 MergePipe 严格执行用户指定的约束。在预算水平相同的情况下，专家数量更多的工作负载会产生更高的专家 I/O，这表明规划决策仍然反映底层问题规模，而非退化为固定启发式方法。

端到端时间反映了预算控制的 I/O。 如图 6(b) 所示，实际运行时间随 I/O 预算单调增加，并紧密呈现出与专家输入/输出相同的趋势。这证实了在此情形下，合并性能受 I/O 限制，且调整预算可以直接、可预测地控制

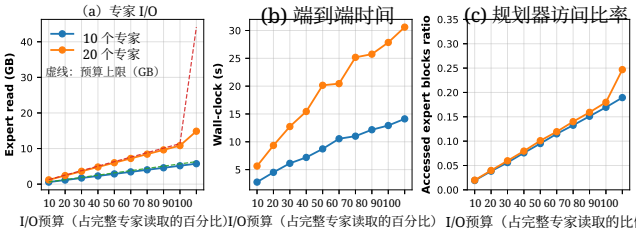


图 6：预算感知规划。(a) 专家读取 I/O 随 I/O 预算单调增长，并受理论预算上限约束。(b) 端到端合并时间与专家 I/O 密切相关。(c) 规划器选取的已访问专家块比例随预算平稳增加。

端到端执行时间。我们观察到，中等范围的预算附近存在轻微的非线性（例如 ~50%），这是由于额外专家块的离散纳入所致，而非系统行为不稳定；超过这些点后，执行时间仍会平稳增长。

规划器访问范围的扩展过程可控且易于理解。图 6(c) 展示了规划器访问的专家参数块比例。已访问块比例随预算近似线性增长，并且在 10 个和 20 个专家的情况下呈现几乎相同的趋势。这表明，规划器的决策主要由预算驱动，且相对于专家总数具有尺度不变性。综合来看，这些结果表明，随着预算增加，MergePipe 的规划器会逐步且可预测地扩大已访问专家集合，从而在 I/O 成本与执行时间之间提供精细且可由用户控制的权衡。

6.4 I3.合并算子之间的通用性

我们评估了 MergePipe 在不同合并算法中的通用性。表4比较了三种代表性算子——平均、TIES 和 DARE——下的朴素执行与 MergePipe，并将专家数量 K 从 2 调整到 20。

在不同算子间保持一致的专家 I/O 控制。 在所有三种算子和各种专家集合大小下，MergePipe 都将总参数 I/O 减少了近一个数量级。朴素执行的总 I/O 随 K 增加而近乎线性增长（例如，在 K=20时超过 170GB），而 MergePipe 将总 I/O 保持在近乎恒定的约 3.5 GB。因此，I/O 比率（Ratio）随 K增加而持续下降，在所有算子下于 K=20 时降至约 2%。这证实了 MergePipe 的 I/O 节省不受合并语义影响，而是源于预算感知的参数访问，而非算子特定优化。

运行时间收益取决于合并稀疏性。 端到端运行时间的改善因算子而异。对于 TIES 和 DARE 等稀疏合并算法，MergePipe 在所有 K下都能实现显著加速。例如，在 TIES 下，MergePipe 在 K=4–8 时将墙钟时间缩短了 90% 以上，在 K=20时仍能带来约 70% 的改善。这些收益源于朴素稀疏合并会反复扫描大型专家检查点，而 MergePipe 则有选择地访问预算限定的参数子集。

相比之下，在非常小的 K（例如 K=2 和 4）时，Average 合并的改进幅度较小，偶尔甚至为负，此时

表4：合并算子的通用性（Llama-3.1-8B）。比较三种合并算子下的朴素执行与 MergePipe。上：总参数输入/输出（MB）以及 MergePipe 所消耗的朴素方法输入/输出比例（Ratio）。下：端到端墙钟时间（秒）和相对改进（Improv. = （$T_{\text{naive}} - T_{\text{MergePipe}}$）/$T_{\text{naive}}$）。

专家 I/O 总量（MB）

	平均			TIES			DARE		
K	朴素方法	MergePipe	比率	朴素方法	MergePipe	比率	朴素方法	MergePipe	比率
2	32127.62	3461.73	10.77%	32127.62	3461.73	10.77%	32127.62	3461.73	**10.77%**
4	44978.67	3461.73	**7.70%**	45766.68	3461.73	**7.56%**	44978.67	3461.73	**7.70%**
6	57829.73	3461.73	**5.99%**	65831.24	3461.73	**5.26%**	65831.24	3461.73	**5.26%**
8	70680.79	3461.73	**4.90%**	79470.30	3461.73	**4.36%**	79470.30	3461.73	**4.36%**
10	83531.85	3461.73	**4.14%**	92321.36	3461.73	**3.75%**	92321.36	3461.73	**3.75%**
12	102808.41	3461.73	**3.37%**	105172.42	3461.73	**3.29%**	105172.42	3461.73	**3.29%**
14	123660.97	3461.73	**2.80%**	126024.99	3461.73	**2.75%**	126024.99	3461.73	**2.75%**
16	137300.04	3461.73	**2.52%**	140046.71	3461.73	**2.47%**	139200.14	3461.73	**2.49%**
18	158152.60	3461.73	**2.19%**	161316.03	3461.73	**2.15%**	160200.60	3461.73	**2.16%**
20	171003.66	3461.73	**2.02%**	174424.50	3461.73	**1.98%**	173200.17	3461.73	**2.00%**

端到端墙钟时间（秒）

	平均			TIES			DARE		
K	朴素方法	MergePipe	改进	朴素方法	MergePipe	改进	朴素方法	MergePipe	改进
2	31.11	41.17	**-32.34%**	404.27	37.17	**90.81%**	79.27	37.78	**52.34%**
4	37.31	50.98	**-36.64%**	491.77	42.31	**91.40%**	78.14	45.40	**41.90%**
6	81.93	56.69	**30.81%**	558.78	46.32	**91.71%**	92.07	43.59	**52.66%**
8	136.17	64.07	**52.95%**	614.27	51.06	**91.69%**	99.10	44.72	**54.87%**
10	159.95	77.36	**51.63%**	578.84	62.89	**89.14%**	101.03	67.22	**33.47%**
12	191.94	84.22	**56.12%**	633.06	80.28	**87.32%**	127.99	84.73	**33.80%**
14	172.67	127.30	**26.28%**	790.65	90.97	**88.49%**	206.89	153.13	**25.98%**
16	195.17	173.52	**11.09%**	941.36	198.65	**78.90%**	248.91	210.03	**15.62%**
18	361.59	334.47	**7.50%**	1051.48	314.52	**70.09%**	395.30	358.85	**9.22%**
20	363.02	338.36	**6.79%**	1203.65	356.32	**70.40%**	464.16	414.44	**10.71%**

朴素执行本身已经很快，规划开销尚未得到摊销。然而，随着 K 增加，MergePipe 始终优于朴素执行，在中等规模下（例如， K=8–12）实现了超过 50% 的运行时间缩减。

案例分析。 在 K=8 下，MergePipe 将所有算子的总 I/O 从 70–80GB 降至 3.5 GB，同时将 TIES 下的墙钟时间从 614s 缩短至 51 s，并将 DARE 下的墙钟时间从 99 s 缩短至 45s。在 K=20 下，尽管合并语义不同，所有算子都表现出相近的 I/O 比率（约 4%），这表明无论专家参数如何组合，MergePipe 的规划器都会执行相同的预算访问模式。总体而言，这些结果表明，MergePipe 能够适用于各种合并算子，始终控制占主导地位的专家 I/O 成本，并在实际的稀疏合并工作负载中带来最大的性能提升。

6.5 I4. 执行开销与系统成本

在本小节中，我们希望回答以下问题："与端到端合并相比，MergePipe 是否会因规划、元数据管理和事务执行而引入不可忽略的系统开销？"我们评估这些组件在实际应用中是否会产生显著开销。

规划开销可以忽略不计。如表5所示，规划仅耗时 1.21 秒，占端到端执行时间的 1.04%

表5：代表性配置（Qwen3-0.6B，16 个专家）下 MergePipe 的执行开销和系统成本。与执行时间和总 I/O 相比，规划和元数据开销较小。

指标	绝对值	相对成本
规划时间（T_{plan}）	1.21 秒	占执行的 1.04%
执行时间（T_{exec}）	116.62 秒	100%
估算的专家 I/O（计划前）	19 842 MB	–
已执行的专家 I/O	23 349 MB	1.18×
总输入/输出量	91 447 MB	100%
元数据目录大小（数据库）	3 468 MB	占总 I/O 的 3.79%
执行清单大小	812 KB	<0.001%

表 6：在固定的专家集合和 I/O 预算下，MergePipe 对块大小的敏感性。块过小会增加开销，而块过大则会降低预算控制粒度。

块大小		16 k	32 k	64 k	128 k	256 k	512 k
TIES	专家 I/O（MB）	4039.55	3520.03	3226.09	3198.74	3390.15	3772.96
	墙钟时间（秒）	77.33	61.50	52.00	45.32	46.33	45.04
DARE	专家 I/O（MB）	4039.55	3520.03	3226.09	3198.74	3390.15	3772.96
	墙钟时间（秒）	83.89	62.47	50.38	46.31	46.33	46.82

。这证实，与合并执行相比，MergePipe 的成本建模、冲突解决和计划生成仅引入极小的开销。在不同专家数量下，我们观察到规划开销同样很小，这表明规划器成本不会随专家数量增加而上升。

规划器引起的 I/O 可通过执行节省的开销得到摊销。 冲突感知规划可能会额外读取专家，以验证候选访问。在此配置下，已执行的专家 I/O 相较于规划前的估计增加了 1.18× 。不过，这一适度开销可通过消除执行过程中的冗余专家扫描得到摊销，因此与朴素合并相比，总 I/O 显著降低。

元数据和事务成本并不高。 持久化元数据（包括 SQLite 目录和谱系信息）仅占总输入/输出量的 3.79%，而执行清单的大小可以忽略不计。与模型参数 I/O 相比，这些成本仍然很低，也不会影响端到端的合并吞吐量。这些结果表明，MergePipe 仅引入少量系统开销，同时有效降低了主要的专家 I/O 成本。规划、元数据管理和事务执行均能良好缩放，不会抵消预算感知合并带来的性能收益。

6.6 I5.消融实验与敏感性分析

在本小节中，我们希望回答："哪些设计选择对 MergePipe 的 I/O 降低至关重要，以及性能对关键系统参数的敏感性如何？"我们研究了 MergePipe 对规划和执行过程中所采用块粒度的敏感性。块大小决定了参数访问的原子单元，并直接影响预算可控性和执行开销。

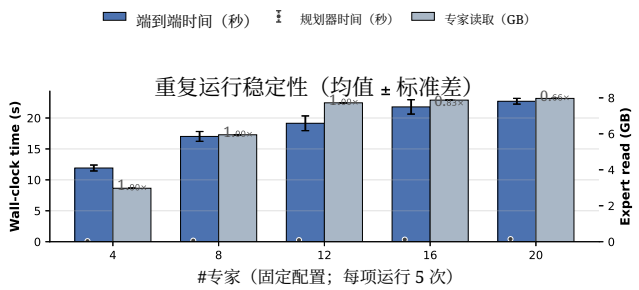


图 7：重复运行稳定性。在固定配置（Qwen3-0.6B、TIES）下进行五次相同的执行，统计均值和标准差。端到端时间和专家输入/输出的方差较低，表明执行行为稳定。

对块大小的敏感性。 表 6 报告了 TIES 和 DARE 算子在不同块大小下的总专家 I/O 量和端到端墙钟时间。对于这两种算子，性能都呈现出清晰且一致的趋势。当块过小时，例如 16 k 或 32 k 时，总 I/O 和执行时间会明显增加。这是由于过度碎片化所致：规划器必须管理和访问大量细粒度块，从而增加元数据查找、索引遍历和 I/O 寻道开销。随着块大小增加，专家 I/O 量和墙钟时间都会下降，并在约 64 k–128 k 时达到最低。在这一范围内，块足够细粒度，能够实现有效的预算感知选择，同时避免过度碎片化。对于过大的块（256 k 及以上），总 I/O 开始再次上升。此时，预算控制变得粗粒度：选择单个大块可能会引入不必要的参数，从而降低选择性专家访问的有效性。尽管如此，墙钟时间仍保持相对稳定，表明 MergePipe 能够平稳退化，而不会表现出病态行为。

不同合并算子下的鲁棒性。 值得注意的是，TIES 和 DARE 在所有块大小下都表现出几乎相同的敏感性曲线。这表明，块大小敏感性主要是一种系统级现象，与合并语义无关。因此，MergePipe 针对块粒度提供了稳定的运行区间，无需进行算子特定的调优。这些结果表明，尽管块大小会影响 MergePipe 的效率，但系统对该参数并不过度敏感。广泛的块大小范围都能实现接近最优的性能，验证了块级规划在实际部署中的鲁棒性。

6.7 重复运行的稳定性

为评估 MergePipe 在相同配置下的稳定性和可重复性，我们开展了重复运行实验：固定模型、合并算子、I/O预算和规划器配置，仅改变专家数量。对于每个专家数量$K \in \{4、8、12、16、20\}$，我们执行相同的合并工作负载**五次**，并报告关键指标的均值和标准差。

总体结果表明，在给定固定计划的情况下，**MergePipe的执行行为是确定性的**，且观测到的方差主要由系统级噪声主导，而非规划不稳定性。图 7 总结了这些结果。端到端墙钟时间随专家数量的增长而平滑且单调地增加，从 K=4 处的约 12秒增至K=20处的22–23秒。在所有设置下，观测到的标准差都很小（通常低于均值的5%），表明在重复执行下端到端性能高度稳定。即使在更大的专家数量下绝对运行时间更高，方差仍然有限，例如在 K=16处围绕21–22秒均值约为 ±1 秒。

表 7：不同预算下的参数偏差与下游性能（Qwen3-0.6B, TIES, K=20）。

	偏差			基准测试		
预算	触及比例	相对 ℓ_2 误差 P95	区块误差	HumanEval	IFEval	DROP
1.0（完整）	0.247	0	0	39.71	68.56	66.81
0.9	0.180	7.23e-4	3.66e-3	42.68	68.82	67.23
0.8	0.159	7.77e-4	3.66e-3	36.59	68.94	67.08
0.7	0.140	8.30e-4	3.98e-3	37.80	68.47	66.26
0.6	0.119	8.30e-4	3.98e-3	37.37	67.87	66.10
0.5	0.101	8.84e-4	3.98e-3	40.24	68.11	66.00

专家读取量更加稳定。如图 7 右侧纵轴所示，在固定 I/O预算下访问的专家参数量几乎是确定的，不同运行之间的方差可以忽略不计。例如，在 K=12、16 和 20 时，专家读取均值分别接近完整专家读取的 1.00×、 0.83× 和 0.66×，多次重复运行间没有明显波动。这证实，MergePipe 的规划器在输入相同的情况下会做出一致的块选择决策，而且专家 I/O 变异不会导致运行时间方差。

规划器开销既小又稳定。值得注意的是，在所有配置下，规划器执行时间均低于 0.2 秒，且标准差非常小。因此，与整体执行时间相比，规划器的变异可以忽略，不会影响端到端稳定性。这表明 MergePipe 在重复运行时能够提供稳定且可重复的 性能。实际观察到的较小运行时间方差，归因于 CPU 调度和后台 I/O 等正常的系统级影响，而非不稳定的规划行为或不一致的专家访问。本节报告的所有实验结果均为五次运行的平均值。

6.8 正确性与质量保持

我们评估预算感知执行是否保留了合并模型的正确性和质量。MergePipe 不会改变合并性能；它只会在预算限制下访问专家参数。

（A）参数级偏差。表 7 比较了相同设置（Qwen 3 0 6 K=20 ℓ_2 $\|\theta_B - \theta_{\text{full}}\|_2 / \|\theta_{\text{full}}\|_2$. B, TIES, ）下预算受限输出 θ_B 与完整读取输出 θ_{full}，并报告相对误差 / 和高百分位块误差。即使预算低至 B=0.5，偏差仍然很小（约为 10^{-3}），并随着访问的专家块减少（触及比例）而平稳增加，表明近似过程受到良好控制。

(B) 下游质量。表 7 报告了三个基准测试上的任务表现，涵盖代码生成（HumanEval [6]）、指令遵循（IFEval [43]）和文本推理（DROP [8]）。在所有预算下，得分均接近全读取基线，且预算降低时没有出现持续性退化。综上，

参数级和任务级的结果表明，在这一代表性配置中，MergePipe 大幅减少了专家输入/输出，同时保留了合并模型的质量。

7 相关工作

大语言模型合并旨在将多个训练好的模型整合为一个模型，无需从头开始重新训练，现已成为现代大语言模型开发流程中的常见组成部分。现有方法大致可分为密集和稀疏合并算子。密集方法，例如权重平均、模型汤 [34],和任务算术（TA） [15],，会对参数进行全局合并，并默认任务更新之间的冲突可以通过平均来解决。相比之下，稀疏合并方法会选择性地纳入特定任务的参数，以减少干扰，包括 TIES [36], 、DARE [40], 以及利用任务向量中的低秩结构的基于奇异值分解的算子 [19, 21]。近期的激活感知方法会根据输入激活动态调整模型权重 [17, 23], ，而混合专家式方法则引入路由机制，在推理时间组合专家 [18, 26]。还有一些研究进一步探讨了模型合并的经验缩放定律和理论性质 [31], ，并有综述对不断发展的相关研究进行了全面总结 [44]。

尽管算法各异，现有合并方法主要关注参数应如何组合，同时默认专家参数的访问成本可以忽略不计。然而，在实际的大语言模型流水线中，反复读取专家检查点会产生大量参数输入/输出开销，且该开销随专家数量线性增长。这项成本既占主导地位，又可以加以控制，但在以往研究中却大多被抽象化处理。MergePipe 通过提供预算感知执行引擎来解决这一独立的瓶颈，该引擎可强制限制专家读取次数，从而在不改变合并语义的前提下实现大规模合并。

大语言模型融合 代表了另一种范式，它通过蒸馏或混合流水线来对齐模型行为，而非直接操控参数。以往研究主要从算法设计以及跨模型的质量迁移角度，探讨在逻辑值、表征或架构组件层面进行对齐 [32] [33],。这些方法与 MergePipe 互为补充：融合方法旨在提升模型性能，而 MergePipe 则专注于在明确的资源约束下，使参数级合并可执行、可预测且具备成本效益；这对于大规模大语言模型系统中的迭代式专家合并至关重要。

机器学习管理系统。 从系统的角度来看，大规模大语言模型开发日益类似于针对大型不可变制品（如检查点、增量和衍生变体）进行的数据管理工作负载，其中索引、复用和数据溯源成为首要需求。此前的系统研究了模型和实验管理抽象，例如用于模型生命周期跟踪和可查询元数据的 ModelDB [12, 30],，以及端到端机器学习平台（如 MLflow），它们记录运行、参数和制品，以提高可复现性和运维可见性 [42]。与此同时，面向机器学习流水线的数据与版本控制工具（如 DVC [2] ）及其他研究 [28, 29] ，着重于数据集与制品版本控制、可重复流水线和可审计差异。工作流引擎和流水线系统 [5, 9] 提供编排与缓存，但通常不会将参数访问成本 作为明确的优化目标。更广泛地说，关于数据沿袭和数据溯源的数据库研究，已为跟踪数据推导过程并支持数据工作流中的事后检查奠定了系统化基础，并开发了实用系统 [4, 10]。

相比之下，MergePipe 与以往的数据溯源和流水线系统互为补充，但关注的是不同的系统级瓶颈。现有的数据溯源系统 [24] 通常将模型检查点视为不透明的制品，因此无法洞察迭代式大语言模型合并的成本结构。因此，它们无法分析或控制此类工作负载中的主导输入/输出模式：反复扫描专家检查点会导致 O(K) 的专家读取成本。MergePipe 则将模型参数视为可管理的数据。它 (i) 将检查点组织成带有持久化统计信息的张量或块，(ii) 将合并计划表示为可复用的执行对象，且 (iii) 在规划和执行期间，明确地将专家参数读取建模为一种受预算约束的资源。这种预算感知、块粒度的执行模型使 MergePipe 有别于以往的流水线和数据溯源工具；后者主要改善可追溯性，却无法为大规模合并提供有界或可预测的专家输入/输出行为。

8 结论

我们介绍了 **MergePipe**，这是一种预算感知执行流水线，将大规模大语言模型合并重新定义为数据管理与执行问题。MergePipe 将专家参数读取 视为主要且可控的成本，采用目录驱动的规划和流式执行，在不依赖特定合并算子的前提下实施明确的专家 I/O 预算。在具有代表性的合并工作负载中，MergePipe 将总 I/O 降低至原来的十分之一，并实现最高 11× 倍的端到端加速（墙钟时间最多缩短 70%）。对于专家数量众多且专家 I/O 占主导地位的离线合并，MergePipe 的优势最为显著；而对于专家集较小或密集型合并，其收益自然会减弱。

大语言模型的使用

在本稿件中，我们仅将大语言模型用作写作辅助工具，包括纠正语法错误、改写以提高可读性，以及进行细微的文风润色。所有实质性分析、实验和结论均为作者原创。此项使用符合当前关于负责任地应用生成式人工智能工具的学术规范。

参考文献

[1] Sandhini Agarwal、Lama Ahmad、Jason Ai、Sam Altman、Andy Applebaum、Edwin Arbus、Rahul K Arora、Yu Bai、Bowen Baker、Haiming Bao 等。2025。gpt-oss-120b 和 gpt-oss-20b 模型卡。*arXiv*预印本 *arXiv:2508.10925*（2025）。[2] Lorena Barreto Simedo Pacheco、Musfiqur Rahman、Fazle Rabbi、Pouya Fathol- lahzadeh、Ahmad Abdellatif、Emad Shihab、Tse-Hsun Chen、Jinqiu Yang 和 Ying Zou。2024。开源机器学习开发中的 DVC：行动与反应。载于 *IEEE/ACM* 人工智能工程 —— 人工智能软件工程第三届国际会议论文集。75–80。[3] Dan Biderman、Jacob Portes、Jose Javier Gonzalez Ortiz、Mansheej Paul、Philip Greengard、Connor Jennings、Daniel King、Sam Havens、Vitaliy Chiley、Jonathan Frankle 等。 [无日期。]LoRA 学得更少，也遗忘得更少。《机器学习研究汇刊》（[无日期]）。[4] Peter Buneman、Sanjeev Khanna 和 Tan Wang-Chiew。2001。为何以及何处：数据溯源的特征描述。载于国际数据库理论会议。Springer，316–330。[5] Marc Bux、Jörgen Brandt、Carsten Lipka、Kamal Hakimzadeh、Jim Dowling 和 Ulf Leser。2015。SAASFEE：可扩展的科学工作流执行引擎。《*VLDB*基金会会刊》8，12（2015），1892–1895。

[6] Mark Chen。2021。对使用代码训练的大语言模型进行评估。arXiv预印本 arXiv:2107.03374（2021）。[7] Jiaxi Cui、Munan Ning、Zongjian Li、Bohua Chen、Yang Yan、Hao Li、Bin Ling、Yonghong Tian 和 Li Yuan。2023。Chatlaw：由知识图谱增强的混合专家大语言模型驱动的多智能体协作法律助手。arXiv预印本 arXiv:2306.16092（2023）。[8] Dheeru Dua、Yizhong Wang、Pradeep Dasigi、Gabriel Stanovsky、Sameer Singh 和 Matt Gardner。2019。DROP：需要对段落进行离散推理的阅读理解基准。载于《2019年北美计算语言学协会分会：人类语言技术会议论文集》，第1卷（长篇与短篇论文）。2368–2378。[9] Sebastian Eggers。2024。自动化数据血缘与流水线提取。《VLDB基金会会刊》。ISSN 2150（2024），8097。[10] Todd J Green、Grigoris Karvounarakis 和 Val Tannen。2007。溯源半环。载于第二十六届 ACM SIGMOD-SIGACT-SIGART 数据库系统原理研讨会论文集。31–40。[11] Daya Guo、Qihao Zhu、Dejian Yang、Zhenda Xie、Kai Dong、Wentao Zhang、Guanting Chen、Xiao Bi、Yu Wu、YK Li 等。2024。DeepSeek-Coder：当大语言模型遇上编程——代码智能的崛起。arXiv预印本 arXiv:2401.14196（2024）。[12] Michael L Hines、Thomas Morse、Michele Migliore、Nicholas T Carnevale 和 Gordon M Shepherd。2004。ModelDB：支持计算神经科学的数据库。《计算神经科学杂志》17，1（2004），7–11。[13] Minghui Hu。2022。集成深度学习：综述。《人工智能工程应用》115（2022），105151。[14] Binyuan Hui、Jian Yang、Zeyu Cui、Jiaxi Yang、Dayiheng Liu、Lei Zhang、Tianyu Liu、Jiajun Zhang、Bowen Yu、Keming Lu 等。2024。Qwen2. 5-coder 技术报告。arXiv预印本 arXiv:2409.12186（2024）。[15] Gabriel Ilharco、Marco Tulio Ribeiro、Mitchell Wortsman、Ludwig Schmidt、Hannaneh Hajishirzi 和 Ali Farhadi。 [无日期。]使用任务算术编辑模型。载于第十一届国际学习表征会议。[16] Aixin Liu、Bei Feng、Bing Xue、Bingxuan Wang、Bochao Wu、Chengda Lu、Cheng- gang Zhao、Chengqi Deng、Chenyu Zhang、Chong Ruan 等。2024。Deepseek-v3 技术报告。arXiv预印本 arXiv:2412.19437（2024）。[17] Shuqi Liu、Han Wu、Bowei He、Xiongwei Han、Mingxuan Yuan 和 Linqi Song。2025。Sens-merging：灵敏度引导的参数平衡，用于合并大语言模型。arXiv预印本 arXiv:2502.12420（2025）。[18] Shuqi Liu、Yuxuan Yao、Bowei He、Zehua Liu、Xiongwei Han、Mingxuan Yuan、Han Wu 和 Linqi Song。2025。1bit-Merging：面向大语言模型的动态量化合并。arXiv预印本 arXiv:2502.10743（2025）。[19] Zehua Liu、Han Wu、Yuxuan Yao、Ruifeng She、Xiongwei Han、Tao Zhong 和 Mingxuan Yuan。2025。LoRE-Merging：探索大语言模型合并中的低秩估计。arXiv预印本 arXiv:2502.10749（2025）。[20] Jinliang Lu、Ziliang Pang、Min Xiao、Yaochen Zhu、Rui Xia 和 Jiajun Zhang。2024。合并、集成与协作！大语言模型时代协作策略综述。arXiv预印本 arXiv:2407.06089（2024）。[21] Zhenyi Lu、Chenghao Fan、Wei Wei、Xiaoye Qu、Dangyang Chen 和 Yu Cheng。2024。Twin-merging：在模型合并中动态整合模块化专业能力。《神经信息处理系统进展》37（2024），78905–78935。[22] Humza Naveed、Asad Ullah Khan、Shi Qiu、Muhammad Saqib、Saeed Anwar、Muhammad Usman、Naveed Akhtar、Nick Barnes 和 Ajmal Mian。2025。大语言模型全面概述。《ACM智能系统与技术汇刊》16，5（2025），1–72。[23] Amin Heyrani Nobari、Kaveh Alim、Ali ArjomandBigdeli、Akash Srivastava、Faez Ahmed 和 Navid Azizan。2025。大语言模型的激活信息引导合并。arXiv预印本 arXiv:2502.02421（2025）。[24] Pingcheng Ruan、Tien Tuan Anh Dinh、Qian Lin、Meihui Zhang、Gang Chen 和 Beng Chin Ooi。2021。LineageChain：面向区块链的细粒度、安全且高效的数据溯源系统。《VLDB期刊》30，1（2021），3–24。[25] Zhihong Shao、Peiyi Wang、Qihao Zhu、Runxin Xu、Junxiao Song、Xiao Bi、Haowei Zhang、Mingchuan Zhang、YK Li、Yang Wu 等。2024。Deepseekmath：推动开放语言模型数学推理能力的极限。arXiv预印本 arXiv:2402.03300（2024）。[26] Sainbayar Sukhbaatar、Olga Golovneva、Vasu Sharma、Hu Xu、Xi Victoria Lin、Baptiste Rozière、Jacob Kahn、Daniel Li、Wen-tau Yih、Jason Weston 等。2024。Branch-train-mix：将专家大语言模型混合到混合专家大语言模型中。arXiv预印本 arXiv:2403.07816（2024）。[27] Yi-Lin Sung、Linjie Li、Kevin Lin、Zhe Gan、Mohit Bansal 和 Lijuan Wang。2023。多模态模型合并的实证研究。载于计算语言学协会论文集：EMNLP 2023 论文选集。1563–1575。[28] Bharath Chandra Vadde 和 VB Munagandla。2024。机器学习时代的 DevOps：弥合开发与数据科学之间的鸿沟。《网络安全与人工智能机器学习研究国际期刊》15，1（2024），530–544。[29] Tom Van Der Weide、Dimitris Papadopoulos、Oleg Smirnov、Michal Zielinski 和 Tim Van Kasteren。2017。端到端机器学习流水线的版本管理。载于第一届端到端机器学习数据管理研讨会论文集。1–9。

[30] Manasi Vartak、Harihar Subramanyam、Wei-En Lee、Srinidhi Viswanathan、Saadiyah Husnoo、Samuel Madden 和 Matei Zaharia。2016。ModelDB：一种机器学习模型管理系统。载于 人在回路数据分析研讨会论文集。1–3。[31] Yuanyi Wang、Yanggan Gu、Yiming Zhang、Qi Zhou、Zhaoyi Yan、Congkai Xie、Xinyao Wang、Jianbo Yuan 和 Hongxia Yang。2025。大语言模型中的模型合并缩放定律。*arXiv* 预印本 *arXiv:2509.24244*（2025）。[32] Yuanyi Wang、Haifeng Sun、Chengsen Wang、Mengde Zhu、Jingyu Wang、Wei Tang、Qi Qi、Zirui Zhuang 和 Jianxin Liao。2024。依赖关系至关重要：多变量时间序列异常检测的图对齐。载于 *2024 IEEE*国际数据挖掘会议（*ICDM*）。IEEE，869–874。[33] Yuanyi Wang、Zhaoyi Yan、Yiming Zhang、Qi Zhou、Yanggan Gu、Fei Wu 和 Hongxia Yang。2025。InfiGFusion：通过高效Gromov-Wasserstein实现基于Logits图的蒸馏，用于模型融合。*arXiv* 预印本 *arXiv:2505.13893*（2025）。[34] Mitchell Wortsman、Gabriel Ilharco、Samir Ya Gadre、Rebecca Roelofs、Raphael Gontijo-Lopes、Ari S Morcos、Hongseok Namkoong、Ali Farhadi、Yair Carmon、Si- mon Kornblith 等。2022。模型汤：对多个微调模型的权重进行平均，可在不增加推理时间的情况下提升准确率。载于 国际机器学习会议。PMLR，23965–23998。[35] Weiwen Xu、Hou Pong Chan、Long Li、Mahani Aljunied、Ruifeng Yuan、Jianyu Wang、Chenghao Xiao、Guizhen Chen、Chaoqun Liu、Zhaodonghui Li 等。2025。灵枢：用于统一多模态医学理解与推理的通用基础模型。*arXiv* 预印本 *arXiv:2506.07044*（2025）。[36] Prateek Yadav、Derek Tam、Leshem Choshen、Colin A Raffel 和 Mohit Bansal。2023。Ties合并：解决模型合并过程中的干扰。神经信息处理系统进展36（2023），7093–7115。[37] An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv 等。2025。Qwen3技术报告。*arXiv* 预印本 *arXiv:2505.09388*（2025）。[38] An Yang、Beichen Zhang、Binyuan Hui、Bofei Gao、Bowen Yu、Chengpeng Li、Dayiheng Liu、Jianhong Tu、Jingren Zhou、Junyang Lin 等。2024。Qwen2. 5-数学技术报告：通过自我改进迈向数学专家模型。*arXiv* 预印本 *arXiv:2409.12122*（2024）。[39] Enneng Yang、Li Shen、Guibing Guo、Xingwei Wang、Xiaochun Cao、Jie Zhang 和 Dacheng Tao。2024。大语言模型、多模态大语言模型及其他模型中的模型合并：方法、理论、应用与机遇。*arXiv* 预印本 *arXiv:2408.07666*（2024）。[40] Le Yu、Bowen Yu、Haiyang Yu、Fei Huang 和 Yongbin Li。2024。语言模型都是超级马力欧：从同源模型中吸收能力这一免费午餐。载于 第四十一届国际机器学习会议。[41] Shengbin Yue、Shujun Liu、Yuxuan Zhou、Chenchen Shen、Siyuan Wang、Yao Xiao、Bingxuan Li、Yun Song、Xiaoyu Shen、Wei Chen 等。2024。LawLLM：具备法律推理和可验证检索的智能法律系统。载于 高级应用数据库系统国际会议。Springer，304–321。[42] Matei Zaharia、Andrew Chen、Aaron Davidson、Ali Ghodsi、Sue Ann Hong、Andy Konwinski、Siddharth Murching、Tomas Nykodym、Paul Ogilvie、Mani Parkhe 等。2018。使用MLflow加速机器学习生命周期。*IEEE*数据工程通报41，4（2018），39–45。[43] Jeffrey Zhou、Tianjian Lu、Swaroop Mishra、Siddhartha Brahma、Sujoy Basu、Yi Luan、Denny Zhou 和 Le Hou。2023。大语言模型的指令遵循评估。*arXiv* 预印本 *arXiv:2311.07911*（2023）。[44] Qi Zhou、Yiming Zhang、Yanggan Gu、Yuanyi Wang、Zhijie Sang、Zhaoyi Yan、Zhen Li、Shengyu Zhang、Fei Wu 和 Hongxia Yang。2025。通过模型融合普及人工智能：综合综述与未来方向。*Nexus*（2025）。