arXiv:2611.32073v1 [cs.DB] 24 Nov 2026

通过感知传递性的大语言模型批量排序实现高效语义 ORDER BY

Akinkunle Akintayo Akinola Arijit Khan美国，鲍灵格林州立大学akinola@bgsu.edu
arijitk@bgsu.edu

摘要—**大语言模型（LLM）使声明式语义排序成为可能：通过 LLM-ORDER BY 算子，根据非结构化数据上的自然语言标准对元组进行排序。高效执行此类算子颇具挑战，因为大语言模型比较成本高、具有随机性，有时还会出现不一致。成对方法需要大量高成本调用。外部归并排序等列表式方法每次调用可对多个项目排序，但它们遵循固定的合并计划，且不会记录先前的答案：每次合并调用都会再次询问那些先前调用已经直接或通过传递性揭示的顺序。我们提出**传递式批量排序 **（TBS），这是一种用于语义排序的感知传递性的物理算子。它会在全局无环偏好图中记录所有已揭示的偏好，并且仅针对相对顺序尚无法推断的项目发出列表式请求。为实现扩展性，TBS 只存储直接观测到的偏好，并通过内存高效可达性索引回答传递性查询；为处理含噪大语言模型输出，它会验证响应并拒绝会导致环的更新。我们使用四种大语言模型，在六个基准数据集上评估 TBS，涵盖结构化排序、产品价格排序、段落排序和个性化推荐。与成对快速排序相比，TBS 将 API 调用次数减少**12–20× **；在相同批大小下，与列表式归并排序相比，它减少 1.1–1.6× 次调用，成本降低** 1.3–5.3× **，同时在大多数基准数据集上达到有竞争力或更高的排序质量；在**亚马逊** 数据集上，它可扩展至一百万种产品。这些结果表明，复用先前推断出的偏好关系，而不是重新查询这些关系，是构建可扩展语义 ORDER BY 算子的有效基础。**

索引词—**语义查询处理；大语言模型引导的排序；大语言模型批量推理；传递闭包**

I. 引言

数据库引擎中的声明式语义排序。将大语言模型（LLM）集成到关系查询引擎中，可以使用主观的自然语言标准，对非结构化属性进行声明式查询 [1], [2]。不同于需要带标签训练数据的监督式学习排序模型 [3], , 或基于预先计算的嵌入索引构建的检索流水线 [4], [5], , 大语言模型可以在查询时以零样本方式理解新的准则。近期系统将这一能力作为扩展 SQL 风格数据流的语义算子提供，包括声明式语义排序子句 LLM-ORDER BY，它根据自然语言准则对元组进行排序 [1], [2]。虽然声明式规范简化了查询编写，但高效执行语义排序十分困难。经典排序算子 [6] 假设比较成本低廉、确定且具有传递性，而 LLM 比较器成本高、具有随机性，并且可能不一致。现有的 LLM-ORDER BY 算子 [1], [2], [7] 通过访问路径选择或基于枢轴的调度来降低成本，但它们缺乏能够保留并复用早期大语言模型调用所揭示偏好的物理算子。

示例 1（基于隐含紧急程度的态势报告优先级排序）：在灾害应急响应期间，运营中心会在关系表中维护实时态势报告：SITREP（id、ward、category、reported_at、description），其中包含数百条持续流入的事件日志。描述属性包含非结构化文本，例如“齐胸深的积水阻断了通往河滨疗养院的道路”，或“停车场有少量积水；无人受伤。”

显式ORDERBY。分析员若要使用标准 SQL 对严重事件进行优先级排序，就必须设计脆弱且经过人工调整的评分启发式方法：

```
SELECT id FROM SITREP ORDER BY 0.15 * recency_
score(reported_at)+ 0.85 * keyword_
hits(description, CRITICAL_TERMS) DESC;
```

此类显式评分函数可能无法捕捉细致的领域语义，并且随着运营状况变化，需要持续进行人工调整。即使使用监督式学习排序模型 [3]来增强此类启发式方法，也需要带标签训练数据和特定任务的特征工程；在快速演变的运营工作负载中，这些都不切实际。

语义ORDERBY。相比之下，声明式语义排序通过自然语言将领域理解工作交由大语言模型处理：

```
SELECT id FROM SITREP LLM-ORDER BY "按即时响应紧迫性排
序，优先考虑生命安全、关键设施通行以及弱势人群。";
```

物理执行瓶颈。尽管概念上简单明了，但在关系查询引擎中执行 LLM-ORDER BY，违背了传统数据库排序所依赖的假设——这些假设 [6], 依赖廉价、确定且严格满足传递性的成对比较器，以实现 $O(n\log\ n)$复杂度。生成式大语言模型比较器违反了所有这些假设：它们成本高昂（每次调用都会产生金钱上的 API 费用和网络延迟）、具有随机性（不同次调用的结果不确定），且不满足传递性（会产生循环偏好）。现有的大语言模型增强排序方法 [2], [8] 存在显著局限：

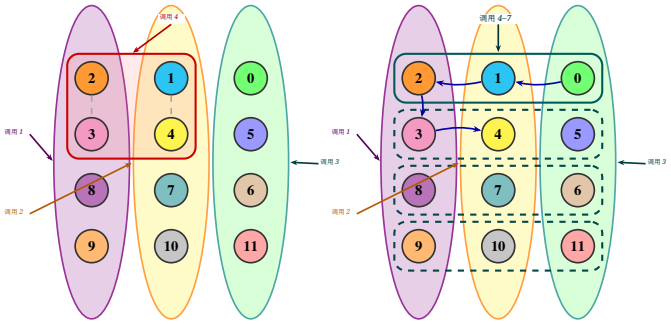
• **高成本的成对比较：**经典排序算法（如快速排序和归并排序）依赖成对比较 [8], [9]。对于比较几乎免费的经典排序算子 [6], , 这尚可接受；但对于大语言模型而言，每次比较都是一次高成本的 API 调用，只能确定一对元素的顺序，尽管一次调用可以为更大的批次排序。因此，比较器成本主导了查询延迟（例如，n = 200 时，快速排序约需 1,500 次调用）。

•**逐点校准：** 逐点评分会为每个元组 [8]分配一个相互独立的标量评分。当每个键都具有模型能够回忆的绝对属性（例如球员的身高）时，这种方法成本低且准确。然而，来自不同提示词的评分彼此之间未经校准，因此在相关性或偏好等相对标准上，逐点评分的表现会下降（§VI-B1）。

•**冗余的列表式合并调用：** 列表式提示通过对大小为 m [2]、 [10]、[11]的批次进行排序来摊销成本。已发表的最强列表式大语言模型 LLM-ORDER BY Schedule 是 OrderByLLM 外部归并排序 [2]: 它先对包含 m 个键的连续运行段进行排序，然后通过大小最多为 m 的进一步列表式调用来合并这些运行段（算法4；合并宽度为来自每个运行段的 $h{=}\lfloor m/2 \rfloor$ 个头）。该 Schedule 从不存储一次调用揭示的成对偏好，因此合并批次可能会再次询问 第1轮已经排序的运行段内成对项——即使合并提示词本身使用了完整的批次预算 m。图1以 $m{=}4$在 $n{=}12$上的情形具体说明了这一点：在三次共享的第1轮调用之后，合并调用4对 $\{2,3,1,4\}$ 进行排序，并重新查询 $2{\prec}3$ 和$1{\prec}4$，而调用1–2已经返回了这些结果；之后的七次合并调用中，每次都至少重复一个已知的运行段内成对项（总计11次调用）。滑动窗口重排序器也存在类似问题：当重叠窗口对相同的键进行重新排序时，就会发生重复。两类方法都无法检测出后续批次已由先前的答案推出。

• **随机噪声与环不一致性：**大语言模型输出经常存在位置偏差、标识符缺失、重复项以及非传递偏好环（$A \succ B \succ C \succ A$）。实用的数据库引擎必须确保面对带噪声且格式错误的生成式回复时，仍满足严格的偏序一致性。

我们的方法：传递式批量排序（TBS）。为应对这些挑战，我们提出了 TBS，这是一种面向语义排序、感知传递性的物理算子。不同于假设比较成本低且结果确定的传统数据库排序算子 [6], ，也不同于每次调用后都会丢弃偏好的 OrderByLLM 归并排序 [2], ，TBS 以成本高昂的列表式大语言模型调用 和持久化偏好有向无环图为核心。它按照固定的三轮批次计划组织键（列排序、递归行排序和对角拼接）；各轮比较的键对大多互不重叠，并将每项已接受的偏好记录为有向无环图边。提交计划中的批次前，TBS 会检查可达性；如果该批次中的所有键对都已能推导出来，就跳过此次调用——这正是图 1 中的 $m{=}4$演示仅需 7 次调用、而归并排序需要 11 次的原因：第 2 轮的行通过传递性闭合全序，无需对角拼接或打包调用。TBS 不会物化二次规模的传递闭包，而是



(a) 列表式归并排序（11 次调用） **(b)** TBS（7 次调用）

图1：使用$m{=}4$ 对乱序输入X=(8, 9, 3, 2, 10, 4, 7, 1, 0, 11, 6, 5)进行演示（升序真实顺序为$0{\prec}1{\prec}\cdots{\prec}11$）。图中绘制的是第1轮之后的 $4{\times}3$ 网格。共享的第1轮（3 次调用）。调用 1 将 $\{8,9,3,2\}$ 排序为 $2{\prec}3{\prec}8{\prec}9$；调用 2 将 $\{10,4,7,1\}$ 排序为 $1{\prec}4{\prec}7{\prec}10$；调用 3 将 $\{0,11,6,5\}$ 排序为$0{\prec}5{\prec}6{\prec}11$。（a）列表式归并排序 [2]（算法 4）随后使用 $h{=}\lfloor m/2 \rfloor{=}2$ 合并这三个已排序序列段（归并批大小最大为 m）。调用 4 将 $\{2,3,1,4\}$（红框）排序为 $1{\prec}2{\prec}3{\prec}4$。虚线框中的 $2{\prec}3$和 $1{\prec}4$ 已由调用 1–2 返回：大小为 4 的归并会再次询问它们。之后还会进行七次归并调用，每次同样会重复至少一个已知的序列段内元素对，总计 11 次大语言模型调用。（b）TBS 将相同的第1轮答案存储在一个有向无环图中。此处第 2 轮对每一行进行排序，如算法2所示（实验默认设置仅排序 R_0，见§V-A）。第 2 轮对四行进行排序（调用 4–7）：$\{2,1,0\}{\rightarrow}0{\prec}1{\prec}2$、 $\{3,4,5\}{\rightarrow}3{\prec}4{\prec}5$、$\{8,7,6\}{\rightarrow}6{\prec}7{\prec}8$、$\{9,10,11\}{\rightarrow}9{\prec}10{\prec}11$（实线框 = 为调用 4；虚线框 = 为调用 5–7）。传递性闭合了完整的链$0{\prec}1{\prec}\cdots{\prec}11$，因此 $u{=}0$：无需对角调用，也无需打包。总计：7 次大语言模型调用（归并排序为 11 次）。

仅存储直接边，并通过动态拓扑序索引查询可达性（内存占用相对于键和已存储偏好为线性）。为处理含噪大语言模型输出，它会通过六项检查验证每条回复，并拒绝会成环的边；为控制成本，当已排序的键对达到目标比例后便提前停止，并自动选择 m。

我们的贡献。我们的贡献如下：

•**感知传递性的物理算子：** 我们对声明式 LLM-ORDER BY 进行形式化，并设计了 TBS——一种三轮批量调度，它将每个列表式答案记录在偏好图中，并跳过其配对已可由传递性推知的批次，从而避免重复询问已知配对；即使归并批大小达到最大值，列表式归并排序仍会重复询问这些配对 m（图 1）。与假设比较器完美无误的经典 k-排序器和网格调度 [12]–[14]，不同，TBS 会跳过已由先前答案确定的批次，并拒绝矛盾。会产生环的更新将被拒绝，因此该图始终无环，且 TBS 始终返回与所有已接受的偏好一致的全序。

• **内存高效可达性索引：**我们用直接边有向无环图和动态拓扑序索引替代显式传递闭包。该索引在使用线性而非二次方内存的同时，能够作出与显式闭包相同的决策；在我们的实验中，其速度提升了 45–1，200×峰值内存减少了 25–83× （预热后的中位缓存重放排序时间；通过 tracemalloc 获取峰值 Python 堆）。

表 I：与先前关于基于大语言模型的排序和语义 ORDER BY 的代表性工作的比较。各列按范围、状态复用以及稳健性与分析分组；✓ 表示该属性存在，–表示该属性缺失或未涉及。Op：针对给定键列表执行数据库语义 ORDER BY。ZS：使用零样本通用聊天模型。Full：具体化全局全序。POG：维护跨调用的已接受偏好的显式全局图。传递性：查询该图以获取传递推导出的关系，并在所有成对关系均已被推导出时跳过一次调用。验证：修复格式错误的输出。批次大小：调整列表式批次大小。成员关系：对批次记录成员关系和提示词顺序进行消融。错误：将排序错误与格式修复分别形式化。

	范围			状态复用	鲁棒性 / 分析				
方法	Op	ZS	Full	POG	Transitivity	Validation	Batch-size	Membership	Errors
OrderByLLM [2]	✓	✓	✓	–	–	✓	–	–	–
LOTUS [1]	✓	✓	✓	–	–	–	–	–	–
ListK [7]	✓	–	✓	–	–	–	–	–	–
LLM-RankFusion [15]	–	✓	–	–	–	–	–	–	–
大语言模型重排序器 [9], [11], [16]	–	✓	–	–	–	–	–	–	–
Setwise [17]	–	✓	–	–	–	–	–	–	–
经过训练的LTR重排序器 [3], [18]	–	–	–	–	–	–	–	–	–
TBS（本文方法）	✓	✓	✓	✓	✓	✓	✓	✓	✓

- **鲁棒性、针对噪声大语言模型的调优与优化：**我们引入了六项检查验证与修复层、环诱导的更新拒绝机制、基于补全内容的提前停止机制，以及基于拐点的批次大小选择方法。我们还研究了第1轮成员预设如何影响质量和成本。
- **全面评估：**在六个基准数据集和四个大语言模型上，TBS比成对式快速排序少调用12–20× 次API，且在每个数据集上，在 m 相当的情况下，调用次数均少于OrderByLLM列表式归并排序（例如，在使用GPT-4o时，*NBA*上为97次对148次，*DL19*上为1,790次对2,347次），同时保持具有竞争力的归并质量，并在 *DL19* 和*SciFact* 上取得领先的平均完整列表nDCG（在 *MovieLens*上与QS_1相差不超过0.008）。

II. 相关工作

本工作属于新兴的AI增强型数据库系统领域，在该领域中，大语言模型被集成到查询执行过程中，以支持对非结构化数据执行语义算子 [1], [2], [7]。我们聚焦于语义大语言模型ORDER BY，其目标是根据自然语言准则，对给定的键列表生成全局排序。既有工作主要在从大语言模型获取排序结果时所采用的交互范式上有所不同，包括逐点式、成对式和列表式方法。我们将回顾这些范式，并讨论其对语义排序的影响。

A. 逐点、成对和列表式范式

传统数据库排序假设比较器 [6]成本低廉、确定且一致。基于大语言模型的比较器违反了这些假设，因为生成过程具有随机性，输出可能取决于呈现顺序 [19], ，而且应用程序接口通常返回排列或成对偏好，而非校准后的分数 [8], [9]。因此，先前的研究探索了逐点式、成对式和列表式排序范式。Dai 等人 [8] 比较了这些方法，并指出没有任何一种范式能在所有任务中占据优势。

在数据库系统中，Zhao 等人 [2] 形式化定义了 LLM-ORDER BY 算子，并研究了多种执行策略，包括成对快速排序（LOTUS [1]）和列表式归并排序调度。这些调度仅通过固定结构（枢轴分区或已排序序列段）复用先前的答案；它们都不会维护跨调用揭示的偏好的显式全局记录，因此无法利用不同批次之间由传递性所蕴含的关系来跳过后续调用。相比之下，我们的工作将揭示的偏好视为共享状态，并利用传递推理减少冗余的大语言模型调用。TBS 进一步设计了三轮网格，使每轮比较的成对对象基本互不重叠，并使用贪心装箱覆盖剩余的未解析对。

ListK [7] 研究使用 RankZephyr [20] 等微调的列表式排序器进行语义排序和 top-K 选择。ListK 专注于专用排序模型和基于枢轴的递归调度，而我们的工作面向通用聊天模型，重点关注通过偏好图维护、感知传递性的查询跳过以及对不一致输出的鲁棒性来执行 LLM-ORDER BY。

SortBench [21] 评估大语言模型对数值、词法和混合排序任务中的列表进行排序的能力。我们的评估通过将语义排序作为数据库算子进行研究，补充了 SortBench，并在结构化排序、段落排序和推荐工作负载中同时衡量排序质量与执行成本。

B. 大语言模型排序的不一致性

大语言模型生成的排序结果往往不一致。Wang 等人指出，GPT-4 在成对比较中存在位置偏差，导致偏好判断不稳定 [19]。Zeng 等人发现，大语言模型生成的排序结果中存在传递性违背，并提出 LLM-RankFusion，通过聚合多个排序器生成的排序结果来提高鲁棒性 [15]。这些工作旨在通过排序聚合和校准来缓解不一致性。相比之下，我们将不一致性视为语义排序算子的系统挑战：检测到的矛盾会被阻止进入维护中的偏好图，从而确保即使单个大语言模型输出存在噪声，存储的排序仍保持无环。

C. 基于大语言模型的排序与重排序

大语言模型越来越多地被用作检索流水线中的重排序器：初始检索阶段会生成一个候选集，随后由大语言模型对其重新排序 [9], [11], [16]。这些方法主要优化 top-K 相关性和检索指标。集合式提示通过每次对少量候选项进行排序，进一步降低推理成本 [17]。相比之下，我们研究的是语义 ORDER BY 的执行，其目标是对一个关系生成完整排序，而不是对一组固定的检索到的文档重新排序。因此，我们的

重点是在保持整个数据集全局排序一致的同时，降低大语言模型比较成本。

D. 学习排序

学习排序（LTR）方法利用带标签的相关性判断或隐式反馈来训练排序函数 [3]，[22]。经典的逐点式、成对式和列表式方法（包括 LambdaMART 和现代交叉编码器架构）在训练数据充足时仍然非常有效 [18], [23]。然而，学习排序假定训练期间已知固定的排序目标。语义 ORDER BY 查询则允许用户通过自然语言动态指定排序标准。因此，我们的目标并非学习一个专用排序器，而是使用通用大语言模型直接执行此类查询，同时尽可能降低比较成本。

E. TBS 的定位

关于基于大语言模型的排序，先前研究涵盖逐点式、成对式和列表式提示，以及用于语义排序的数据库算子；这些方法都在排序质量与推理成本之间权衡。如表 I 所示，能够针对给定键列表返回全序的系统（OrderByLLM、LOTUS、ListK）不会显式维护由先前调用揭示的偏好全局记录；而解决排序不一致性或质量问题的方法（ LLM-RankFusion、重排序器、Set-wise）则不针对数据库 ORDER BY。在表中的基于大语言模型的方法中，TBS 是唯一一种将全局全序与持续维护的偏好图相结合，并通过传递性跳过调用的方法。传递性复用已应用于众包排序 [24]；我们的贡献是将其扩展至百万级键规模的列表式大语言模型调用。更多相关工作见附录。

因此，TBS 解决了一个互补的执行问题：在列表式调用之间复用偏好，以避免重复的大语言模型评估。它将已接受的偏好记录在一个有向无环图中，利用可达性识别已解析的比较，并且仅存储直接边以减少内存占用。它还会验证格式错误回复并拒绝导致环的更新；我们还研究了批大小和批次成员关系如何影响质量和成本。由于该图由大语言模型输出构建，复用的收益取决于已接受偏好的准确性，这使质量—成本权衡成为评估的核心。

III. 预备知识

大语言模型排序算子。我们对一种基于大语言模型的排序算子 LLM-order by 进行建模：该算子接收一组 n键 X 和一个自然语言排序准则 c，并返回与 c一致的排序列表，即经过确定性平局决策后的全序 π 。键可以表示任意对象，例如数据库行标识符、文档 ID 或命名实体。准则以自由文本形式表达（例如“按身高升序排列；如有并列，则按字母顺序排列”或“按段落与查询的相关性排序”）。在示例 1 中，键是报告 ID 值，而准则是自然语言的紧急程度规范，用于 LLM-order by。该算子只能通过生成式接口访问：给定任意子集$B \subseteq X$ 和准则，大语言模型会返回一个有序列表，其中恰好包含 B 中的所有元素。

从批量输出中提取成对关系。对于批大小 $m \geq 2$，设 $B = \{b_1, \ldots, b_m\}$ 是一个键批次。大语言模型返回 B 的排列 $\sigma(B) = (x_1, \ldots\ldots, x_m)$，我们将其解释为严格成对关系的集合$x_i \succ x_j \quad \forall 1 \leq i < j \leq m$，即由所列序列隐含的关于 B 的全序。这一解释统一适用于各项任务：在基于值的排序中，它表示数值或序数比较；在段落排序中，它表示相对于某个查询的成对相关性偏好。

维护的关系集、闭包与一致性。该算法维护一个由有序对 $\mathcal{R}$ (a, b) 构成的集合，每个有序对表示在该准则下 a 严格优于 b。每次批次响应后，所有蕴含出的成对关系都会被添加到 $\mathcal{R}$，并立即对该集合取传递闭包。未来查询中会跳过 $\mathcal{R}$已蕴含的任何关系。

插入新的有序对 $x(x, y)$ 时，我们使用自反集合 $A = \{x\} \cup \{a : a \succ x\}$ 和 $S = \{y\} \cup \{b : y \succ b\}$ 传播其传递后果（等价于 §IV-A 中的 $A = \{x\} \cup$ Greater_than(x) 和 $S = \{y\} \cup$Less_than(y)），从而推断出 $A \times S$ 中的每一对。如果插入会与现有关系矛盾，即形成有向环，则拒绝该插入，以确保 $\mathcal{R}$保持严格偏序。尽管三轮调度通过网格布局固定了批次成员关系，但在每次列表式调用之前，我们都会检查预定批次中的每一对是否已通过闭包解析；若是，则跳过此次调用。当某个批次因仍包含未解析对而被查询时，模型可能会返回与 $\mathcal{R}$ 中已有关系矛盾的排序；此类边会在插入前被丢弃。

图视图和增量索引。 我们将 $\mathcal{R}$ 表示为有向图 $G = (V, E)$，其中 V 是键的集合，边 $u \to v$ 表示 $(u, v) \in \mathcal{R}$。由于会导致环的更新会被拒绝，因此 G 保持无环，从而编码了严格偏序。为支持高效更新，每个键 x 都维护两个索引集合：

$$\texttt{Less\_than}(x) = \{\, y : (x, y) \in \mathcal{R} \,\},$$
$$\texttt{Greater\_than}(x) = \{\, y : (y, x) \in \mathcal{R} \,\}.$$

当新的关系 (a, b) 被接受时，我们沿着 a 的前驱节点和 b 的后继节点更新这些集合，从而纳入此次插入所产生的所有传递性后果。

提取全序。令 in(x) = |Greater_than(x)| 表示 x 在 G中的入度。我们通过拓扑遍历（例如 Kahn 算法 [25]）来获得一个线性序：重复选择一个满足 in(x) = 0 的顶点，将其追加到输出中，在更新度数时将其移除，并将其出邻居的入度减一。当多个顶点的入度为零时，我们应用固定的确定性平局规则（例如字典序），从而选出 G 的一个线性扩展（使用最小堆得到字典序最小的拓扑

序）。只有当 G本身已经是全序时，该线性扩展才是唯一的。

问题陈述。我们将本文所解决的物理执行任务形式化。

问题 1（传递性感知的批量 LLM 排序）：输入：一个键列表 $X = [x_1, \ldots, x_n]$、一个自然语言排序准则 c、一个批大小 $m \geq 2$, 以及一个列表式排序器 LLM Sort。对于任意批次 $B \subseteq X$ （其中包含 $2 \leq |B| \leq m$），该排序器返回 B 的排列 $\sigma(B)$。

输出：对 X 的确定性全序 π 。

交互：算法选择批次 B_1、.....、$B_T(|B_t| \leq m)$，通过查询这些批次来收集 LLM Sort上的输出 $\sigma(B_t)$，并将所得成对关系插入关系图 G ，同时应用传递闭包。对于其中的成对关系已被解析的批次，将跳过不再处理；而任何会在 G 中引入环的边都会被排除。当 Schedule 终止时，使用 Kahn 拓扑排序从 G 中提取 π，并在不可比较的键之间采用字典序打破平局。

目标：在确保 π 与任务参考顺序高度一致的同时，尽量减少列表式调用次数 T （全序任务使用 Kendall's τ，段落排序使用 nDCG；见 §VI）。成对排序（例如使用大语言模型比较器的快速排序 QS_1）需要 $\Theta(n\log\ n)$ 次调用，每次确定一对元素的顺序。对 m 个键进行一次列表式调用，最多可同时排列 $\binom{m}{2}$ 对元素，因此我们的三轮批量调度（§IV）需要 $O(\frac{n}{m}\log_m n)$ 次调用，达到列表式排序的下界（§IV-A5）；基于传递性的跳过还能进一步减少实际调用次数。

IV. 方法

本节分三部分介绍传递式批量排序（TBS）：完美列表式预言机下的算法（§IV-A）；避免显式传递闭包所需二次方内存的直接边可达性索引（§IV-B）；以及将原始 LLM 响应转换为排列的验证层（§IV-C）。第 V 节介绍了 TBS 实际部署中的高成本效益启发式优化。

A. 传递式批量排序

1) 预言机与共享状态：对于任意批次 B 及 $2 \leq |B| \leq m$，完美预言机会返回 B在准则 c下的真实排列。TBS 维护三个供每个过程读取和更新的结构：

1) 关系图 $G = (V,\ E)$，保持传递闭包；2) 每个键的未决邻域，即在 $\mathcal{R}$ 中，与 x 的相对顺序尚未确定的键集合（当 $x \succ y$ 和 $y \succ x$ 均未知时，我们记作 $x? y$ ），

$$U[x] \ = \ \{\, y \neq x : x \,?\, y \,\}.$$

3) 未决无序对的数量，

$$u \ = \ \tfrac{1}{2} \sum_{x \in X} |U[x]|.$$

初始时，对于所有 $x \in X$， $E = \emptyset$、 $U[x] = X \setminus \{x\}$ 以及 $u = \binom{n}{2}$。

从逻辑上讲， G 是全序，当且仅当 $u = 0$。传递闭包图 G、集合 $U[\cdot]$和计数器 u 仅用于

算法1完美预言机传递式批量排序

输入： 带有 $n = |X|$的键集合 X 、排序准则 c、批大小 $m \geq 2$**输出：** 在 X 上的全序 π **共享状态：** 在 $V = X$ 上的关系图 $G = (V, E)$ 、未解决邻域 $U[\cdot]$以及未解析对计数器 u，由算法2和算法3读取并更新

```
G ← (V = X,  E = ∅)
U[x] ← X \ {x}          对所有 x ∈ X：
u ← (n 2)
：Schedule(X) {三轮递归调度}5:
Pack() {解析剩余未解析对}6: 返回 π ←拓扑排序(G)
```

作为该算法的逻辑 规范，因为显式物化它们需要 $\Theta(n^2)$ 存储空间，在大规模场景下难以承受。实际中它们从不被物化；而是根据稀疏直接边有向无环图（§IV-B）按需推导，从而使内存开销与 n 及已存储偏好的数量呈线性关系。因此，测试 $u = 0$ 在维护计数器的显式闭包下为 $O(1)$，但在实现中不再是 $O(1)$，其中 u 通过分块列扫描恢复。

2）批次网格：对于键列表 X 和 $k = \lceil |X|/m \rceil$，TBS 将 X 划分为列 C_0、.....、C_{k-1}，每列最多包含 m 个连续键。令 $C_j[r]$ 为 C_j的第 r 个键。行 r为 $R_r = \{C_j[r] : 0 \leq j < k,\ r < |C_j|\}$。对角线 d（$d = 0$、.....、$m+k-2$，但如果最后一列不完整，则可能更少）为 $D_d = \{\, C_j[r] \mid r + j = d,\ 0 \leq j < k,\ r < |C_j| \,\}$ $|D_d| \leq$ m k ，因此 min(,)。少于两个键的对角线不包含可供比较的键对，因此绝不会被查询。这包括 $D_0 = \{C_0[0]\}$ ，以及最后一列完整时的 $D_{m+k-2} = \{C_{k-1}[m-1]\}$。

理解该网格的一种有效方式是按成对比较族来看：列覆盖列内比较，行在相同秩上跨列传播顺序，而对角线则覆盖前两轮之后剩余的跨秩边界交互。

3）算法：TBS 由三个在共享状态（关系图 G、未解决邻域 $U[\cdot]$和计数器 u）上运行的算法组成。算法1是主要过程：在完美预言机下，它先在 X 上运行一次算法2的三轮调度，然后调用 PACK 一次，对剩余的未解析对进行排序；此后， $u = 0$和 G 的拓扑序都是唯一的。算法3定义了发出预言机调用所用的两个基本操作：RESOLVE 对一个批次进行排序，只需单次预言机调用（如果批次中的每一对都已排序，则跳过），并将结果合并到 G 中；PACK 则为尚未解析的对构建批次，并将其发送给 RESOLVE。

给定键列表 X，如果 $|X| \leq m$，Schedule（算法2）只会发起一次解析调用。否则，它会将 X 拆分为$k = \lceil |X|/m \rceil$ 列，并分三轮进行。第*1*轮解析每一列，并使其保持排序顺序，从而得到k 条不相交链。第 *2* 轮 从已排序的列中取出第 r个键，组成每一行 R_r ，并按照相同的 G、 $U[\cdot]$和 u 对其递归调度。由于 $|R_r| \leq k < |X|$，递归会终止。由于每个已排序的列都是一条链

算法2 Schedule(L)：键列表 L上的三轮调度

输入： 键列表 L
使用： 批大小 m 和共享状态 G、 $U[\cdot]$、 u
作用： 更新 G、 $U[\cdot]$和 u

如果 $u = 0$ **或** $|L| < 2$，**则**
  返回
如果 $|L| \le m$，**则**
  解析(L)
  返回
$k \leftarrow \lceil |L|/m \rceil$
将 L 拆分为多列 C_0、……、 C_{k-1}，每列大小至多为 m
第1轮：列排序
对于 $j = 0$ **至** $k-1$，**执行**
  解析(C_j)；将 C_j 保持为排序顺序
第 2 轮：带递归的行排序
对于 $r = 0$ **至** $m-1$ **do**
  $R_r \leftarrow \{C_j[r] : 0 \le j < k, \ \ r < |C_j|\}$
  Schedule(R_r) $\{|R_r| \le \lceil |L|/m \rceil < |L|$ 对于 $|L| > m\}$
第3轮：对角拼接
对于 $d = 1$ **到** $m + k - 3$，**执行**
  如果 $u = 0$，**则**
    返回
  $D_d \leftarrow \{C_j[r] : 0 \le j < k, \ 0 \le r < |C_j|, \ r + j = d\}$
  如果 $|D_d| \ge 2$，**则**
    解析(D_d)

算法3 预言机调用原语：解析和 Pack

过程解析(B)
如果 $|B| < 2$ **则**
  返回
如果存在 a, $b \in B$ 使得 $b \in U[a]$ **则**
  $\sigma \leftarrow$ Oracle_Sort(B, c) {批次 B 在准则 c}下的真实顺序
  将 σ 所蕴含的所有成对关系插入 G
  根据传递性增量式更新 G
  对于每个新解析出的无序对 $\{x, y\}$，从 $U[x]$ 中移除 y 并从 $U[y]$ 中移除 x
  更新 u

过程Pack()
当 $u > 0$ **时**
  $b \leftarrow$ 最大参数$_{x \in X}$ $|U[x]|$
  $B \leftarrow \{b\}$ {使用最高未解析度的键进行初始化}
  当 $|B| < m$ **时**
    $s \leftarrow$ 最大值$_{c \notin B}$ $|U[c] \cap B|$
    如果 $s = 0$ **那么**
      中断
    $d \leftarrow$ 最大参数$_{c \notin B}$ $|U[c] \cap B|$
    $B \leftarrow B \cup \{d\}$
  解析(B)

其中位置 0 最高，若一行的结果为 $x \succ y$，且满足 $x = C_i[r]$ 和 $y = C_j[r]$，则根据传递性， C_i 中位于 x 及以上的每个键都先于 C_j中位于 y 及以下的每个键。这会产生 $(r+1)(|C_j| - r)$ 个跨列对；对所有 r 而言，当 $|C_j| = m$ 时，其数量至多为 $\binom{m+1}{2}^2$。第*3*轮解析对角线 $D_1, \ldots, D_{m+k-3}$。对角线上的键分别位于不同的行和列，因此第1至第2轮不会直接比较它们；每次对角线调用都会跨越列边界，将相邻的行连接起来。只要 $u = 0$，Schedule 就会立即返回。

Resolve(B) 首先检查 $U[\cdot]$ 在批次 B中是否存在未解析对；如果没有，则不会发出调用，因此已通过传递性确定的批次无需任何成本。否则，一次预言机调用会返回 $\sigma(B) = (x_1, \ldots, x_{|B|})$，其边$x_i \succ x_j$ 会针对 $i < j$ 逐条合并。如果 $b \in$Less_than(a)，则会跳过 边$a \succ b$，因为它已经确定；如果 $a \in$Less_than(b)，则会拒绝 该边，因为它会形成一个环；在预言机下不会发生拒绝，而在大语言模型下，该边会被丢弃并计为一次矛盾。否则，令 $\mathcal{A} = \{a\} \cup$Greater_than(a) 且 $\mathcal{S} = \{b\} \cup$ Less_than(b)。根据传递性，每个尚未存在于 E 中的节点对 $(a', \ b') \in \mathcal{A} \times \mathcal{S}$ 都会被插入， a' 和 b' 会从彼此的邻域中移除，并且 u 会针对每一对递减一次。由于每个无序对从 $U[\cdot]$ 中移除的次数至多为一次，因此在整个运行过程中，移除 操作的摊销成本为 $O(n^2)$。然而，定位 $\mathcal{A} \times \mathcal{S}$ 中的哪些节点对是新增的，每次合并都需要成本$O(|\mathcal{A}|\,|\mathcal{S}|)$（每次调用在最坏情况下可达 $O(n^2)$），因此，若没有一种只枚举新增节点对的数据结构，总扫描成本可能达到 $O(T\,n^2)$；§IV-B 介绍了实践中采用的高效直接边实现。

对角阶段不必解析每一对跨组元素。例如，如果没有任何批次同时包含两个键，那么关系 $a \succ b$、 $a \succ c$、 $b \succ d$ 和 $c \succ d$ 就无法确定 b 和 c 的顺序。这类剩余比较正是记录在未解决邻域 $U[\cdot]$ 中的内容。为了处理它们，Pack 会贪心地构建每个批次：先将未解决度最高的键放入批次，然后反复加入与当前批次共享最多未解决对的键。因此，每个打包批次都至少包含一个未解决对，所以在完美预言机下，每次调用 Resolve 都会严格减少 u，从而保证终止。由于 Pack 由 $U[\cdot]$驱动，它只检查剩余的未解决对，不会重新扫描整个 Full $\binom{n}{2}$ 对空间。

4) 正确性： 在预言机下，每条已接受的边都是正确的，且没有插入会遭到拒绝，因此 G 仍然是真实顺序的传递闭包子图。由于只有在 $u = 0$ 时 Pack 才会终止，每一对键最终都会排定顺序，因此对 G 进行拓扑排序即可恢复真实全序（$\tau = 1$）。对于含噪比较器，成环边会被拒绝，从而保持 G 的无环性。因此，输出始终是已接受偏好的有效线性扩展，剩余不可比较的键则按确定性的字典序决胜规则排序。

5) 成本： 我们将算子成本定义为列表式大语言模型调用的次数，每个批量请求均计为一次调用，无论其大小如何（$2 \le |B| \le m$）。由于调用长度各不相同，我们还会报告每次运行的总令牌数和美元成本（§VI-A4）。忽略基于传递性的跳过，SCHEDULE（算法2，所有行）在大小为 n 的输入上的调用复杂度满足

$$T(n) \le m\,T\Big(\Big\lceil \frac{n}{m} \Big\rceil\Big) + O\Big(\frac{n}{m} + m\Big),$$

其中，第1轮最多执行 $\lceil n/m \rceil$ 列调用；第 2 轮递归地对 m 行进行排序，每行最多包含 $\lceil n/m \rceil$个元素；第3轮最多评估 $\lceil n/m \rceil + m - 1$ 个对角批次。递归的第i 层包含 m^i 个大小为 n/m^i的子问题，每个子问题执行 $O(n/m^{i+1}+m)$次调用，因此第 i层共执行 $O(n/m+m^{i+1})$次调用。对 $\log_m n$ 层求和，第一项为$O\big(\frac{n}{m}\log_m n\big)$，第二项是一个由最深层主导的几何级数，为 $O(n)$。因此

$$T(n) = O\Big(\frac{n}{m}\log_m n + n\Big).$$

每次对 m 个键的调用最多揭示所需 $\log_2 n!$ 位信息中的 $\log_2 m!$ 位，因此任何列表式方案都至少需要 $\Omega(\log n!/\log m!)$，即 $=\Omega((n/m)\log_m n)$ 次调用；SCHEDULE 的首项与此界相符。该界仅针对 SCHEDULE；能够保证完整性的 PACK 还会增加少量调用次数。

B. 内存高效可达性索引

显式传递闭包（§IV-A）存储每个已知对和每个开放对，因此无论进展如何，其内存占用都是 $\Theta(n^2)$；合并一条边 $a \succ b$ 时，需要为 a 的每个祖先与 b 的每个后代插入一对（包括它们自身，即 a 和 b），每次合并最多可插入 $n^2/4$ 对[1]。相反，我们仅将已接受的答案存储为稀疏有向无环图，并按需推导所有闭包量，因此内存占用取决于答案数量，而非 n^2；每次插入只会涉及图中受影响的部分。该索引仅改变实现方式：它发出相同的批次，接受和拒绝相同的边，并生成与显式闭包相同的输出 π。

1) 直接边：令 $H = (X, E_d)$ 仅包含从列表式答案中接受的边；若 H 存在一条从 x 到 y、长度至少为一的有向路径，则记为 $x \rightsquigarrow y$ [2]。传递闭包G 随后定义为 $x \succ_G y \iff x \rightsquigarrow y$。

Resolve(B) 按固定顺序检查 $\sigma(B) = (x_1, \ldots, x_{|B|})$ 中的各对：先检查相邻对 (x_i, x_{i+1})，再检查满足 $j \geq i+2$ 的 (x_i, x_j)。如果满足 $a \succ_G b$，则跳过 (a, b)；如果满足 $b \succ_G a$，则将其作为矛盾拒绝；否则将其追加到 E_d。如果没有相邻对被拒绝，那么路径 $x_1 \to \cdots \to x_{|B|}$ 位于 G 中，因此所有剩余的对都已蕴含其中，无需测试即可跳过。否则，通过剪枝搜索各标记一次每个 x_i 的后代和祖先，并在每次插入后更新标记；这样得到的结果与逐对测试完全相同。由于是否接受仅取决于可达性，而显式闭包存储的恰好就是这种关系，因此两种实现会作出完全相同的判断。内存占用为 $O(n + |E_d|)$，其中 E_d 对每个答案中的每个相邻对最多保存一条边，另外还保存少量非相邻插入项。

2) 动态拓扑序：为了在不存储闭包的情况下回答可达性查询，我们维护 H 的拓扑序，以限定每次查询的搜索区域，并在每次插入后进行局部修复。具体而言，我们维护一个双射 ord：$X \to \{0, \ldots, n-1\}$，使得 $(a, b) \in E_d \implies \mathrm{ord}(a) < \mathrm{ord}(b)$，并在每次插入后使用 Pearce 和 Kelly 的算法 [27] 对其进行修复。插入 (a, b) 且 $\mathrm{ord}(a) < \mathrm{ord}(b)$ 时，无需进行更改。否则，只对受影响区域重新排序；在此定义中，$\rightsquigarrow$ 被视为自反的：

$$\delta_F = \{v : b \rightsquigarrow v,\ \mathrm{ord}(v) < \mathrm{ord}(a)\},$$
$$\delta_B = \{v : v \rightsquigarrow a,\ \mathrm{ord}(v) > \mathrm{ord}(b)\}.$$

[1]令 $\mathcal{A}$ 为 a 及其祖先的集合，令 $\mathcal{S}$ 为 b 及其后代的集合，因此一次合并会插入 $|\mathcal{A}| \cdot |\mathcal{S}|$ 对。这两个集合不相交：如果某个键同时属于二者，就会得到 $b \rightsquigarrow a$，该键会在插入前被拒绝。因此 $|\mathcal{A}| + |\mathcal{S}| \leq n$，根据算术—几何平均不等式，有 [26]、$|\mathcal{A}| \cdot |\mathcal{S}| \leq \binom{|\mathcal{A}|+|\mathcal{S}|}{2}^2 \leq n^2/4$。[2]零长度路径会得到 $x \rightsquigarrow x$，从而得到 $x \succ_G x$，即某个键的排名高于其自身。

将 $\mathrm{ord}(\delta_B \cup \delta_F)$ 的位置汇集起来并重新分配，先分配给 δ_B，再分配给 δ_F，两者都保持各自在原有顺序中的相对顺序。这两个集合互不相交，因为在插入前已验证 $b \not\succ_G a$。成本为 $O\big(\sum_{v \in \delta_B \cup \delta_F}(1 + \deg v) + |\delta| \log |\delta|\big)$，其中 $\delta = \delta_B \cup \delta_F$，其范围仅限于该区域。任何有效的初始顺序都会得出相同的答案，因此可以根据启发式排序为 ord 设定初始值，以减少重新排序。

3) 剪枝可达性查询：</style}从 x 到 y 的路径上的每个顶点都在 ord(orequals y) 中排在 y 之前，因此

$$x \succ_G y \iff \mathrm{ord}(x) < \mathrm{ord}(y) \ \wedge\ y \in \mathrm{Reach}_H(x; S_y),$$

其中 $S_y = \{v : \mathrm{ord}(v) < \mathrm{ord}(y)\} \cup \{y\}$，而 $\mathrm{Reach}_H(x; S)$ 是从 x 开始进行深度优先搜索、且仅扩展 S中顶点所到达的顶点集合。如果 $\mathrm{ord}(x) > \mathrm{ord}(y)$，则查询无需进行任何搜索即可返回 false。

链快捷方式。第1轮通过一次列表式调用对每列 C_j 进行排序，因此每列都是 G中的一条链：它包含最密集的已知关系块（$\binom{|C_j|}{2}$ 对），但只需存储 $|C_j|-1$ 条边。各列彼此不相交，并且在任何其他调用之前完成排序，因此它们的答案不会相互矛盾 G，且会被全部接受。我们利用这一结构，采用固定的链覆盖：对每个键 x，存储其所在列 ch(x)及其在该列中的位置 pos(x)（位置 0 表示排名最高的键），总计需要 $2n$ 个整数。按链和位置为顶点标注，并通过逐链比较来回答查询，是压缩传递闭包 [28], [29]的一种经典技术。在我们的场景中，构建覆盖无需任何成本：第1轮列已经是排好序的链，因此无需计算链分解；并且随着边的插入，覆盖始终保持固定。

Theorem 1. 两种方式提1) 同一链。如果 ch(x) =ch(y)，列顺序无需任何搜索即可确定查询结果：$x \succ_G y \Leftrightarrow$ pos(x) <pos(y)。2) 提前终止。如果从 x 开始的搜索到达顶点v，且 ch(v) = ch(y) 和 pos(v) ≤pos(y)，那么根据链顺序，$v = y$或 $v \succ_G y$。结合 $x \rightsquigarrow v$，传递性可得 $x \succ_G y$，因此搜索以肯定答案终止，无需定位 y 本身。

这两条规则都会缩短搜索过程，且不会改变其结果。

批次跳过测试。在发起预言机调用之前，Resolve(B)会检查 B 是否仍包含未解析对（算法3，第4行）。显式闭包可以通过扫描 $U[\cdot]$来回答这一问题。由于此处未物化$U[\cdot]$，我们改为检查 $\{a, b\} \subseteq B$ 中的每一对是否已在 G中确定顺序。因为 $x \succ_G y$要求 $\mathrm{ord}(x) < \mathrm{ord}(y)$，每对只需沿一个方向查询，即与 ord 一致的方向，并可应用链快捷方式。检查会在遇到第一个两个方向都未确定顺序的对时停止，因此通常会提前终止。如果所有对的顺序都已确定，则可零成本跳过该批次。

按阻塞列扫描精确计数。调度器会反复需要两个统计量。未解决度$d(x) = |U[x]|$ 用于由 Pack 为每个批次选取未解决度最高的键作为种子（算法3，Pack，第3行）。未解析对总数 u用于 Schedule 的终止测试（算法2，第1行和第17行）以及

Pack 的循环条件（算法3，Pack，第2行）。在显式闭包表述下，这些量可以直接从维护的祖先集合和后代集合中获得。然而，在直接边实现中，传递闭包不会被显式构造。因此，我们通过对有向无环图 $H = (X, E_d)$进行分块列扫描，直接计算相同的计数。回想一下，第1轮列 $C_0, \ldots, C_{w-1}$ 构成了 H的链覆盖。对于一组列，我们按拓扑序进行反向扫描。对于每个键 x 和列c，我们维护一个临时标签

$$\lambda[x][c] = \min\Big(\{pos(x)+1 : ch(x)=c\} \cup \{\lambda[y][c] : (x,y)\in E_d\} \cup \{pos(y) : (x,y)\in E_d,\ ch(y)=c\}\Big),$$

其中 $\min\emptyset = +\infty$。中间集合会从每个 后继 y传播 $\lambda[y][c]$ ，无论其位于哪一列 y ；只有直接位置项由 $ch(y) = c$进行保护。该标签标识列 c 中已知可从 x 到达的第一个位置：

$$\lambda[x][c] = \min\{p : x \succ_G C_c[p]\}.$$

由于链中 x 的后代构成该链的一个后缀，因此 x的后代数量为

$$|D(x)| = \sum_{c=0}^{w-1} \max\big(0, |C_c| - \lambda[x][c]\big).$$

一次对称的正向扫描计算 $|A(x)|$。随后即可直接得到未解决度：

$$d(x) = (n-1) - |D(x)| - |A(x)|.$$

对所有键求和 $|D(x)|$ ，即可得到有序对的总数；未解析对的数量则是剩余的补集。因此，阻塞扫描无需物化传递闭包，即可计算调度器所需的精确统计量。

完备性测试。判断 G 是否为全序无需计数。由于 ord 是 H 的拓扑序， G 当且仅当每一对连续键之间都是直接边：

$$\textsc{Complete}(G) \Leftrightarrow (\mathrm{ord}^{-1}(i), \mathrm{ord}^{-1}(i{+}1)) \in E_d \ \ \forall\, i < n{-}1.$$

事实上，如果 G是全序，则连续键构成覆盖对，而覆盖对必须由有向边连接；反过来，所列边构成一条哈密顿路径，这意味着所有键对之间都存在关系。该测试在最坏情况下的成本为$O(n + |E_d|)$，通常会在第一个缺失的边处失败。

4）打包与输出：Pack 使用通过一轮正向与反向扫描得到的最大参数$_x$ $d(x)$，为每个批次打包种子。要扩展批次 B，需要计算每个候选项 c的覆盖增益 $|U[c] \cap B|$ ，并通过增量方式维护：将b' 添加到 B后，从 b' 开始进行正向和反向搜索，标记其后代和祖先 $D(b') \cup A(b')$，即已与 b'排序的键，而每个未标记的候选项的覆盖增益都会增加 1。每个批次的成本为 $O\big(m(n + |E_d|)\big)$ 。如遇平局，则按字典序键顺序打破。最终顺序通过在 H 上运行 Kahn算法，并使用以字典序排名为键的最小堆，耗时为 $O\big((n + |E_d|)\log\ n\big)$ 。当某个键在 G中的所有祖先都已输出后，该键便恰好处于就绪状态，因此输出结果等于 G 的拓扑序，并按字典序打破平局。

表II：大语言模型列表式回复的格式验证检查。

检查项	要求	典型违规
解析	$\lvert L\rvert \geq 1$	空回复或无法解析的回复
最小长度	$\lvert L\rvert \geq 2$ （如果 $\lvert B\rvert \geq 2$）	单项列表
完整性	$B \subseteq L$	遗漏的批次键
无额外项	$L \subseteq B$	臆造项
无重复项	每个键出现一次	重复键
项目匹配	不区分大小写地匹配 B	大小写或格式不匹配

C. LLM输出验证与修复

列表式回复不一定是所请求批次的排列。合并之前，TBS 会将回复解析为列表 L，通过与批次 B进行不区分大小写的匹配来规范化项目标识符，并验证 L是否为 B的排列（表II）。如果任何检查未通过， L将进行修复：仅保留回复和 B中都出现的键，并按回复中的顺序排列（每个键仅保留首次出现），同时删除与 B不匹配的令牌。对于遗漏键的回复，系统会重新询问，最多尝试五次。如果回复仍不完整，则按呈现顺序追加缺失的键；在我们的运行中，从未触发此回退机制。如果剩余的有效键少于两个，则丢弃该回复，且不合并此次调用产生的任何边。因此，修复会保留模型对其实际排序的键所给出的相对顺序，并且只贡献这些边。

验证仅处理格式错误。格式正确但与偏好图 G 相矛盾的回复仍可通过验证，而会在合并阶段通过拒绝诱发环的边来处理。

V. 优化策略

算法1假设预言机完美，并持续运行，直到每一对元素都已排序（$u = 0$）。使用大语言模型时，每次调用的成本都很高，回复也可能存在噪声，因此不一定值得完全解决所有 $\binom{n}{2}$ 对。本节介绍在算法1之上采用的启发式方法，以便 (i) 减少调用次数，(ii) 控制哪些键共同进入一个批次，以及 (iii) 选择批大小 m。这些方法会改变所发出的批次以及 Schedule 何时停止，但不会改变如何将回复合并为 G。因此， G仍保持无环，且 π 仍是 G 的线性扩展（§IV-A）。这些方法不属于预言机正确性论证，也可能无法得到 $u > 0$。

A. 降低成本的执行规则

第2轮行排序选项。算法2会递归地对每一行进行排序。为节省调用次数，第 2 轮也可以只对首行 R_0 进行排序，既可以采用递归方式，也可以使用最多包含 m 键的重叠窗口。随后，更深层的行将通过对角拼接和传递性间接地与其余部分建立关联。在我们的实验中，递归 R_0-only 是默认设置；对每一行进行排序（算法2）则作为消融实验进行评估（表V）。

对角线批处理。较短的对角线包含的键较少。对其中一条对角线进行一次调用只会使用少量批处理容量。因此，我们会将连续的较短对角线合并为一个最多包含 m 键的批次，而较长的对角线则单独处理。

提前停止。令 u_t 表示第 t个第3轮批次结束后尚未解析的对的数量，$\rho_t = 1 - u_t/\binom{n}{2}$ 表示有序对所占比例。第3轮（先处理对角线，再执行 Pack）在以下三个条件之一满足时立即停止发出调用：补全内容($u_t = 0$)；停滞，即连续执行固定数量的批次都没有新增有序对；或 θ-补全内容，$\rho_t \geq \theta$ 针对目标 $\theta \in (0, 1]$。如果在处理对角线期间达到 $\rho_t \geq \theta$，则跳过剩余对角线；否则 Pack 会继续执行，直到达到该条件。停止操作不会插入任何边，因此 π 仍是 G的线性扩展。剩余的 u_t 个未解析对按照第III节中的确定性平局决策排序，因此停止操作只会影响这些对的顺序。

未解析对打包。在预言机下，算法1总是以 Pack 结束，这是满足 $\tau = 1$所必需的。在大语言模型下，Pack 是一个可选开关。它会对剩余对额外发出调用，而根据带噪声的回复对这些对排序可能会引入矛盾。当完整性比成本更重要时，我们会启用它；否则依赖提前停止。

B. 批次成员关系与提示词顺序

列表式质量不仅取决于 m，还取决于哪些键被分到同一批次，以及它们在提示词 [10], [19]中的出现顺序。在 TBS 中，成员关系由 X 划分为列的第 1 轮分区 $\{C_j\}$固定。行（第 2 轮）和对角线（第3轮）均由同一分区推导得出，因此单个 成员关系预设 便可控制整个批次网格。我们考虑四种预设：

- **连续**：按输入顺序将 X 拆分为大小为 m的连续片段（输入在排序前会随机打乱一次，因此这已经是一个随机分区）；
- **随机**：使用第二个随机种子重新打乱 X，然后再进行拆分；将其与连续分区进行比较，可以衡量两个随机分区之间的结果差异；
- **相似度**：按基于词元的文本相似度对键进行聚类，并将各簇打包到大小为 m的列中；
- **嵌入相似度划分**：打包方式相同，但聚类依据的是嵌入相似度。

在每个提示词中，键均按分区顺序列出。哪个预设效果最佳取决于任务和模型；我们将在实验中对此进行研究。

C. 批次大小选择

批大小 m需要在大语言模型API调用次数与质量之间权衡。较小的 m会产生更多列、行和对角线，因此调用次数也更多。较大的 m 则会给上下文窗口带来压力，并降低列表式准确率。因此，我们通过一次性扫描，从候选集 $\mathcal{M}$ 中为每个模型和任务选择m。对于每个 $m \in \mathcal{M}$，我们运行一次 TBS，并记录调用总次数 c_m 和质量得分 q_m（Kendall's τ或 nDCG）。这些点（c_m，q_m）按调用次数排序，并重新编号为（x_i，y_i）、$i = 0, \ldots N-1$，其中 $N = |\mathcal{M}|$。随后，在该模型和任务的后续运行中固定所选的 m。扫描需要一个参考顺序来计算 q_m；无需标签的估计留待未来工作。

几何拐点。对于 $N \geq 3$，令 P_0 和 P_{N-1} 分别为调用次数最小点和调用次数最大点。拐点 $i^\star$ 是内部点中到弦 $\overline{P_0 P_{N-1}}$ [30]的垂直距离最大的点。

端点感知效用。我们将调用次数和质量归一化为$[0, 1]$，

$$\tilde{x}_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}, \qquad \tilde{y}_i = \frac{y_i - y_{\min}}{y_{\max} - y_{\min}},$$
$$U_i = \alpha\, \tilde{y}_i + (1 - \alpha)\,(1 - \tilde{x}_i), \qquad \alpha \in [0, 1],$$

其中，较小的 α 有利于减少调用次数。我们使用 $\alpha = 0.7$，对质量赋予比调用次数节省更高的权重。如果某个端点（$i = 0$或 $i = N-1$）的效用严格高于拐点，则选择该端点的批次大小。否则选择 $m_{i^\star}$。

VI. 实验结果

我们评估**传递式批量排序（TBS）**能否利用传递性，在批量查询间复用偏好，从而改善基于大语言模型的排序的质量—成本权衡。我们将 TBS 与成对式、逐点式和列表式基线方法进行比较；量化利用传递性感知优化带来的 API 调用节省；并分析关键设计选择如何影响排序质量和成本。为评估其稳健性和通用性，我们在多个大语言模型提供商以及两类基准上评估TBS：全序恢复（*NBA*球员身高、世界人口）和基于相关性的排序（*DL19*、*Sci-Fact*、*MovieLens*）。**我们的代码和数据可在以下网址获取：**
https://github.com/kakinola23/TBS-LLM-OrderBy。

研究问题。
RQ1 质量—成本权衡。TBS 能否在显著减少大语言模型 API 调用的同时，达到或超过基线方法的排序质量（Kendall τ、nDCG）？**RQ2 优化的贡献**。各设计组件（通过跳过查询进行传递性维护、可达性索引、第 2 轮行排序启发式和第 3 轮提前终止）分别在减少 API 调用和保持排序质量方面有多大贡献？**RQ3 跨模型和任务的稳健性**。不同大语言模型提供商以及全序和基于相关性的基准之间，质量和成本有何差异？**RQ4 批次大小敏感性**。批次大小 m 与所提出的拐点/端点选择策略共同如何影响质量和成本？**RQ5 批次策略**。第1轮成员预设（连续、随机、相似度、嵌入相似度划分）如何影响排序质量和 API 调用次数？

A. 实验设置

1) 数据集与任务：我们使用六个基准数据集：两个带有数值型真实标注、需要模型从参数化知识中回忆信息的列表（*NBA*球员身高和世界人口）；大规模商品价格排序（亚马逊）；

查询驱动的段落排序（*DL19*）；主张–文档排序（*SciFact*）；以及基于用户画像的电影排序（*MovieLens*）。

*a) NBA*球员身高：列表包含 200 名身高经过验证的 NBA 球员 [31]。按照 Zhao 等人 [2] 的做法，X 仅包含球员姓名，而 c 要求按身高升序排列，身高相同时按字母顺序排列，因此大语言模型必须依赖参数化知识。真实排序按（身高，姓名）排序。

b) 世界人口：我们使用 OrderByLLM 基准数据集 [2]: 包含 200 个国家和地区的 2020 年人口数据。与 NBA 球员身高一样，X 仅包含名称，因此大语言模型必须依赖参数化知识；c 要求按人口升序排列，人口相同时按字母顺序排列。真实排序按（人口，名称）排序，并使用整个列表的 Kendall τ 衡量质量。

c) 亚马逊价格排序：我们从 AmazonReviews2023 产品元数据 [32]，中抽取 n=1，000，000 个有价格的产品，每个父级ASIN（亚马逊为产品及其变体分配的标识符）保留一行，价格范围为 [\$0.50，\$10,000]，且标题至少包含三个词。每个键都是一个附有标示价格的产品标题，因此比较几乎不会出错。因此，我们使用此基准数据集测试 TBS 的簿记开销和调用量能否扩展到 10^6 个键，而非衡量语义质量。准则 c 要求按价格升序排列，价格相同时按标题字母顺序排列。**金标准**。标示价格决定全序（价格相同时依次按标题和 ASIN 排序），我们根据该排序计算 Kendall τ。可扩展性测试使用完整的 1M 列表；较小规模的测试则从同一语料库中抽取固定的随机样本（例如，n=20K）。

*d) DL19*段落排序：我们采用 Zhao 等人 [2]的 DL19 协议：43 个 TREC 2019深度学习查询，每个查询均配有 BM25 检索得到的 MS MARCO v1 段落前 100 条。对于每个查询，我们根据段落与查询文本的相关性，对其 100 个段落进行全序排列，该相关性作为准则 c；键为显示了完整文本的段落ID。我们报告这 43 个查询的完整列表 nDCG 均值。**金标准**。与 TREC 一样，人工评审员提供 0 到 3 的分级相关性标签（相关性判断文件），其中 3 表示最相关。

*e) SciFact*主张排序：我们采用 LOTUS [1] 的 SciFact排序协议，对 300 个 SciFact测试主张 [33],中的前 50 个固定切片进行评测；每个主张均配有通过 E5-base-v2 检索得到的前 100 篇摘要。对于每个主张，我们根据摘要与主张文本的相关性，对其 100 篇摘要进行全序排列，该相关性作为准则 c；键为显示了文本的摘要ID。我们报告这 50 个主张的完整列表 nDCG 均值。**金标准**。SciFact 提供二元相关性标签（相关性判断文件，0/1），用于指示摘要是否与主张相关。

f) MovieLens 个性化排序：MovieLens [34]也被用于比较逐点式、成对式和列表式大语言模型提示词 [8]。我们固定选取 63 位至少有 70 条评分的用户，并从每位用户评分过的电影中抽取 n=70 部进行排序。准则 c 是一段简短的自然语言用户画像；键为电影标识符，并显示电影名称和类型。**金标准**。用户评分（1–5）作为完整列表 nDCG 的分级增益，我们报告 63 位用户的平均 nDCG。

2）算法与基线：

a) 所提方法：我们使用大语言模型替代预言机运行 TBS（算法 1–3），并采用六项检查验证（§IV-C）。表 III 列出了我们研究的变体。

表III：命名的 TBS 变体。

Name	Axis	设置／变更
TBS	默认	第1轮；递归 R_0 第 2 轮；对角线，然后 Pack (θ=1.0)；传递性推理；可达性索引
TBS-NoReachIndex	索引	不使用可达性索引（繁重的全对 bookkeeping）
TBS-AllRows	第 2 轮	递归地对每一行进行排序 (算法2)
TBS-Window	第 2 轮	对 R_0 （重叠大小为 $\leq m$ 的窗口）进行窗口排序
TBS-EarlyStop	第3轮	当 $\rho_t \geq \theta$ (θ=0.9 时停止对角线启用时）
TBS-Pack	Pack	启用未解决的打包 Schedule
TBS-NoTrans	传递性	仅直接边：不跳过已解析内容批次，不进行环拒绝（RQ2 消融）

b) 基线方法：我们将 TBS 与 §II-A（表 I）中的排序范式进行比较，每种范式均由一种既有系统或提示设计实现（如下所列）。我们重新实现每种基线方法，并在我们的每个数据集上运行，确保使用与 TBS相同的大语言模型、温度参数、乱序输入、评估键和指标。因此，在每张对比表中，TBS 与所有基线方法使用同一个大语言模型（RQ1 中为 GPT-4o；报告三个模型时为 GPT-4.1、GPT-4o 和 GPT-4o-mini）。

表 I 还列出了我们未作为基线方法运行的其他方法，因为它们针对的是不同问题。LOTUS [1] 和 ListK [7] 是语义 top-K 算子；LOTUS 的完整排序路径采用成对快速排序，我们将其作为 QS_1纳入比较。LLM- RankFusion [15] 会聚合多个排序结果，而经过训练的学习排序重排器 [3], [18] 需要带标签训练数据。滑动窗口列表式重排器 [11], [16] 和集合式提示 [17] 也能生成完整排序，但它们旨在对检索候选项进行 top-K 重排。因此，在列表式调度中，我们将目标为完整 ORDER BY 的 OrderByLLM 归并排序作为比较对象。成对排序提示 [9]则由 QS_1代表。

- **成对快速排序**（OrderByLLM QS_1 [2]）：使用大语言模型成对比较器 [8], [9]的快速排序。每次比较都是一次大语言模型调用，用于确定单个键对的顺序，因此排序需要进行 $O(n \log n)$ 次调用。
- **逐点评分**（n调用次数）：每个键使用一个提示词，请大语言模型给出数值评分（例如预测身高或相关性）；回复会被解析为浮点数，然后按照 c [2]、[8]所要求的方向，根据评分对键进行排序。例如，如果大语言模型分别为*PlayerA*、*PlayerB*和*PlayerC*返回 2.01、1.96 和 2.11，则身高升序为 *PlayerB* $\prec$*PlayerA* $\prec$ *PlayerC*。
- **OrderByLLM 列表式归并排序** [2]：一种批处理外部归并排序，先对连续的 m 键运行排序，然后通过进一步的列表式调用合并这些运行段。与 TBS 不同，它不会复用早期调用揭示的关系。在每项实验中，我们使用与 TBS 相同的批大小 m 运行该方法。

3）模型：我们使用四个大语言模型评估 TBS：GPT-4.1、GPT-4o 和 GPT-4o-mini。GPT-5-nano 仅用于亚马逊价格排序 [3]。除 GPT-5-nano 外，所有调用的温度参数均设为 0；GPT-5-nano 的 API 仅接受默认温度。

4）指标：

a）质量：对于*NBA*、世界人口和亚马逊（每个都是由 n 个键组成的单一列表），我们报告预测排名 π 与真实排名 $\pi^\star$之间的 Kendall τ_a：

$$\tau_a(\pi,\pi^\star)=\frac{C-D}{\binom{n}{2}},$$

其中，C（分别为 D）表示在 π 和 $\pi^\star$下相对顺序一致（分别为不一致）的键对数量，而 τ_a=1表示完全匹配。由于金标准并列项按名称或标题打破，因此两种顺序都是全序，无需进行并列校正。

对于*DL19*、*SciFact*和*MovieLens*，我们报告逐实例全列表归一化折损累计增益（nDCG）的均值。给定分级增益 g_i（*DL19*：TREC 0–3；*SciFact*：二元 0/1；*MovieLens*：评分 1–5）和预测排名 r_i，

$$\mathrm{DCG}=\sum_{i=1}^{n}\frac{g_i}{\log_2(r_i+1)},\qquad \mathrm{nDCG}=\frac{\mathrm{DCG}}{\mathrm{IDCG}},$$

其中，理想折损累计增益（IDCG）是按 g_i计算的理想排名对应的 DCG。我们对每个基准数据集中的实例取 nDCG 平均值：43 个*DL19* 查询的 nDCG_Q、50 条*SciFact* 主张的 nDCG_C，以及 63 位*MovieLens*用户的 nDCG_U。

b) 成本：我们报告三个成本指标：(i) 每次排序的大语言模型调用次数（在多实例数据集中，按每个查询、主张或用户计算），不包括可选的稳健性检查；(ii) 总令牌数（提示词加补全内容，由提供商返回）；以及 (iii) 按模型公布的标准费率估算的美元成本。调用次数和美元成本反映不同的成本，因此分别报告：调用次数受 API 速率限制和单请求开销限制，而美元成本取决于令牌数，因此像逐点评分这样使用许多简短提示词的方法，调用次数可能多于 TBS，但成本反而更低。

5) 实现：TBS 使用 Python 3 实现。所有批次调用均采用统一的列表式提示词布局和结构化输出，遵循 Zhao 等人 [2]的协议；各任务的差异仅在于键的呈现方式（名称、段落或主张文本、电影标题和类型，或 亚马逊标题及价格）。每个提示词还会说明其金标准顺序的字母顺序平局决胜规则。

a) 批大小：我们通过 §V-C 中的过程，从候选集 $\mathcal{M}=\{4,\dots,14\}$ 中为每个模型和数据集选择 m。对于 GPT-4o，参数搜索在每个数据集上都选择了 m=10，因此所有 RQ1 比较均使用 m=10；在 RQ3 中，每个模型使用各自的 m，因为较小的模型每次调用时能可靠排序的项更少。对于目标为百万项列表的亚马逊，我们在 n=200下搜索更大的候选值 $\mathcal{M}_{\text{Amazon}}=\{10,20,40,75\}$，并选择 m=40：m=20 和 m=40 在质量和成本上持平，而较大的值会使第1轮列的数量减半，达到n=10^6。启用提前停止时，我们使用 θ=0.9，并将第3轮停滞上限设为 2；其他所有运行均使用 θ=1.0。

[3]由于提示词中已显示价格，该任务不需要事实回忆，仅测试 TBS 的记账和调用次数是否能够扩展到 10^6 个键；使用小型模型即可，并能使百万键运行的成本保持在可承受范围内

表 IV：RQ1：QS_1,逐点式、合并和 TBS（GPT-4o，m=10）。TBS 使用每个数据集在第 2 轮中的最佳设置。亚马逊**使用** n=1，000。

数据集	指标	QS_1	PW	合并	TBS
NBA	τ ↑	**0.943**	0.938	0.879	0.876
	调用次数 ↓	1,483	200	148	**97**
	令牌数（百万）↓	0.49	**0.05**	0.09	0.06
	成本（美元）↓	3.07	**0.34**	0.64	0.43
人口	τ ↑	0.976	**0.986**	0.933	0.944
	调用次数 ↓	1,421	200	152	**121**
	令牌数（百万）↓	0.37	**0.04**	0.08	0.07
	成本（美元）↓	2.01	**0.21**	0.58	0.45
亚马逊	τ ↑	1.000	**1.000**	0.999	0.972
	调用次数↓	10,351	1,000	1,051	**643**
	令牌数（百万）↓	3.53	**0.25**	0.90	0.52
	成本（美元）↓	16.92	**1.21**	5.10	2.91
DL19	nDCG_Q ↑	0.907	0.855	0.916	**0.921**
	调用次数↓	30,913	4,300	2,347	**1,790**
	令牌数（百万）↓	15.59	**1.62**	3.33	2.24
	成本（美元）↓	79.21	**6.28**	15.43	10.49
SciFact	nDCG_C ↑	0.849	0.691	0.850	**0.854**
	调用次数↓	44,047	5,000	2,511	**2,200**
	令牌数（百万）↓	42.11	**3.07**	9.22	6.98
	成本（美元）↓	169.59	**10.83**	33.14	25.48
MovieLens	nDCG_U ↑	**0.887**	0.870	0.866	0.879
	调用次数↓	24,424	4,410	2,141	**1,832**
	令牌数（百万）↓	7.76	2.34	2.26	**0.87**
	成本（美元）↓	20.03	11.35	13.83	**2.61**

b) 基线方法：所有基线方法均在与 TBS 相同的大语言模型、温度参数、乱序输入和评估键设置下重新实现，归并排序使用相同的批大小 m。

B. 结果

由于篇幅限制，此处报告 RQ1 和 RQ2 的结果；RQ3–RQ5 的结果见附录。

1) RQ1：质量—成本权衡：按照 §VI-A2 中的协议，我们在六个数据集上使用单个大语言模型（GPT-4o），并采用表 V 中各数据集表现最佳的第 2 轮设置，将 TBS 与成对式、逐点式和列表式基线进行比较。表 IV 报告了排序质量和执行成本。

a) 成对快速排序$_1$：与成对快速排序相比，TBS 始终能大幅降低 API 调用次数、令牌数量和货币支出。差距源于 TBS 通过传递推理复用偏好，而 QS_1 则独立解析各项比较。质量方面，结果不一。QS_1 在基于回忆的排序任务上仍表现更强，因为模型往往能够直接根据记忆中的事实回答单项比较。然而，TBS 在*DL19* 和*SciFact*上优于 QS_1，在*MovieLens* 上则略逊一筹（0。879 对比 0。887）。这表明，当排序需要细致的比较性判断而非事实回忆时，跨列表式批次聚合证据往往更为有效。

b) 逐点评分：逐点评分在*NBA*、人口和亚马逊 排序任务中具有竞争力，因为每个项目都有一个可以估计的潜在绝对属性

表 V：RQ2 消融实验：Q(质量)表示 τ 在*NBA*/人口/亚马逊 上的均值，以及*DL19*/*SciFact*/*MovieLens*上的完整列表 nDCG；# 为大语言模型调用次数；Tokens (Tok) 以 M 为单位。(a) NoTrans 与 TBS（GPT-4o-mini）的比较。(b) 亚马逊 上的 ReachIndex (RI) 与显式闭包的比较（GPT-5-nano，m=40）；两个后端的 τ/#/Tok/$ 均匹配；峰值内存（mem.）以 MB 为单位；20K/1M 仅使用 RI 和 θ=0.9。(c) θ=0.90 与完全打包的比较（GPT-4o；亚马逊$_{1k}$：GPT-5-nano）。(d) Round 2 R_0/all-rows ×递归（推荐）/窗口（胜出）。

(a) 传递性

	NoTrans				TBS			
数据集	Q↑	#↓	Tok↓	$↓	Q↑	#↓	Tok↓	$↓
NBA (m=4)	0.354	3,436	1.18	0.41	**0.395**	**306**	**0.09**	**0.02**
流行度 (m=6)	0.946	1,532	0.64	0.25	**0.952**	**202**	**0.08**	**0.03**
DL19 (m=5)	**0.915**	24,469	19.03	4.71	0.900	**4,522**	**3.40**	**0.85**
SciFact(m=4)	0.841	44,545	70.56	14.72	**0.871**	**7,392**	**11.48**	**2.42**
机器学习(m=10)	0.848	5,859	3.06	0.59	**0.878**	**1,784**	**0.86**	**0.16**

(b) 可达性（亚马逊）

	共享				时间（秒）↓		峰值 MB↓	
n	τ↑	#↓	Tok↓	$↓	NoRI	RI	NoRI	RI
200	0.9999	57	0.13	0.04	2.19	**0.05**	2.5	**0.1**
500	0.9997	94	0.39	0.13	34.8	**0.40**	17.0	**0.6**
1,000	0.999	171	0.96	0.33	495.1	**0.41**	49.8	**0.6**
20K	0.946	1,421	11.18	3.78	–	–	–	–
1M	0.923	91,383	829.45	286.37	–	–	–	–

(c) 提前停止（θ=0.90**vs. Full**）

	θ=0.90				Full			
数据集	Q↑	#↓	Tok↓	$↓	Q↑	#↓	Tok↓	$↓
NBA	0.851	**64**	0.04	**0.28**	**0.876**	97	0.06	0.43
人口	0.867	**67**	0.04	**0.24**	**0.944**	121	0.07	0.45
亚马逊$_{1k}$	0.949	**102**	0.50	**0.17**	**0.9997**	171	0.96	0.33
DL19	0.885	**1,279**	1.64	**7.71**	**0.921**	1,790	2.24	10.49
SciFact	0.528	**1,469**	4.64	**17.01**	**0.854**	2,200	6.98	25.48
MovieLens	0.876	**1,320**	0.63	**1.90**	**0.879**	1,832	0.87	2.61

(d) 第 2 轮（R_0/**All-rows** × **推荐/胜出**）

	仅R_0				所有行			
	Rec.		Win.		Rec.		Win.	
数据集	Q↑	#↓	Q↑	#↓	Q↑	#↓	Q↑	#↓
NBA	**0.876**	97	0.852	**94**	0.841	108	0.861	103
人口	0.932	101	0.926	**98**	**0.944**	121	0.927	109
DL19	**0.921**	**1,790**	**0.921**	**1,790**	0.911	1,847	0.911	1,847
SciFact	0.839	**2,102**	0.839	**2,102**	**0.854**	2,200	**0.854**	2,200
MovieLens	**0.879**	**1,832**	**0.879**	**1,832**	0.870	2,085	0.870	2,085
亚马逊$_{1k}$	**0.999**	171	**0.999**	171	**0.999**	**165**	**0.999**	**165**

，可以独立进行估计。在此类场景中，为每个键分配一个分数既简单又准确。在相关性和偏好排序任务中，这一优势基本消失，因为质量取决于相对比较，而非绝对值。在这些任务中，TBS始终优于逐点评分，这表明列表式推理能够捕捉上下文关系，而独立评分提示词无法可靠地对其进行建模。

c) 列表式归并排序：与归并排序的比较能够隔离传递性复用的影响，因为这两种方法都依赖列表式提示。跨数据集来看，TBS 在实现相当或更高排序质量的同时，所需的大语言模型交互更少，整体成本也更低。这一优势源于维护偏好图，并避免处理那些排序关系已由先前答案隐含确定的批次；而归并排序则会按照固定的合并计划反复处理已经确立的关系。

d) 总结：结果揭示了明显的任务依赖型权衡。当排序准则对应可独立估算的数值属性（如身高或人口）时，逐点评分是最优选择。相比之下，对于语义相关性和偏好排序，TBS 始终更有效，因为在这些任务中，相对比较比绝对分数更重要。与成对式和列表式基线相比，TBS 通过复用先前获取的偏好，而非反复查询，大幅减少了大语言模型调用次数，同时达到相当的质量水平。总体而言，结果支持了 TBS 的核心前提：传递性感知偏好复用能够显著提升效率，同时保持甚至提高语义排序任务的排序质量。

2) RQ2：优化的贡献：我们每次从默认配置（§VI-A5）中移除一个开关，进行消融实验。除非另有说明，运行均使用 GPT-4o、m=10 以及表 IV 中的提示词和输入。表 V 总结了全部四项消融实验。

传递性。TBS-NoTrans 不跳过任何批次，并持续打包，直到观察到每一对（表 Va；GPT-4o-mini 在其 RQ3 m 上）。在每个数据集上，传递性都将调用次数减少了 3–11× 。质量参差不齐：NoTrans 仅在 *DL19* 上领先；TBS 在其余四个数据集上领先，因此在含噪比较器下，主要优势仍然是成本。

可达性索引。在亚马逊（GPT-5-nano、m=40）上，两个后端都会重放相同的答案，因此 τ、调用次数、令牌数和 $ 均相同（表 Vb）。ReachIndex 的速度提升了 45–1，200×，内存用量减少了 25–83× ，从而可实现 n=20K（τ=0.946）和 n=1M（τ=0.923，$286；$\theta$=0.9）。

提前停止。在 $\rho_t \geq \theta$ 时停止第 3 轮可节省 28–45% 的调用次数（θ=0.90）（表 Vc）。*NBA*/*MovieLens* 的质量仍接近 Full；*SciFact*则大幅下降，因为每个主张通常仍有一个相关摘要未得到解析。

第 2 轮。仅使用递归 R_0在*NBA*、*DL19*、*MovieLens*和亚马逊 上以最少或接近最少的调用次数取得最佳或并列最佳结果（表 Vd）；在*NBA*/人口上，窗口还可进一步减少少量调用次数。All-rows 仅在人口（+0.012）和*SciFact*（+0.015）上有所帮助，但调用次数增加了 5–20%。令牌和美元成本与调用次数同步变化。

要点。传递性复用是降低成本的主要手段；可达性索引支持 10^6 个键；递归 R_0 是可靠的默认方案；当相关性分散在许多键上时，θ<1 是合理的选择。

VII. 结论

我们提出了传递式批量排序（TBS），这是一种列表式 ORDER BY 运算符，它在有向无环图中记录偏好，并跳过已经由传递性推断出的成对比较。直接边可达性索引使内存保持线性，并让 TBS 能够扩展到 亚马逊上的 10^6 个键。在使用 GPT-4o 基线方法的六个数据集上，TBS 比成对快速排序少调用 12–20× 次，比列表式归并排序少调用 1.1–1.6× 次；在四个数据集上，其质量高于归并排序（在*NBA* 上低 0.003，在 亚马逊上低 0.027），并且在 *DL19* 和*SciFact*上所有方法中质量最高。在没有绝对数值尺度且预算受限的排序任务中，TBS 表现最佳；而在*NBA*、*Population*和亚马逊上，逐点评分仍然更强，在*NBA*、*Population*、亚马逊和*MovieLens*上 QS_1 更强。未来工作包括考虑传递性的前K选择、基于成本的访问路径选择，以及在无需遍历搜索的情况下设置最优m和 θ 。

参考文献

[1] L. Patel、S. Jha、M. Pan、H. Gupta、P. Asawa、C. Guestrin 和 M. Zaharia，“语义算子：丰富的、基于人工智能的数据处理的声明式模型”，2024 年。[2] F. Zhao、J. Chen、Y. Pan、T. Rabbani、Sohaib、D. Agrawal、A. El Abbadi、P. Aggarwal、A. Datta 和 D. Tsirogiannis，“使用大语言模型实现高效排序的访问路径”，2025 年，代码作为 OrderByLLM基准测试发布。[3] T.-Y. Liu，《信息检索的学习排序》。Springer，2011 年。[4] V. Karpukhin、B. Oguz、S. Min、P. Lewis、L. Wu、S. Edunov、D. Chen 和 W. Yih，“面向开放域问答的密集段落检索”，载于《自然语言处理实证方法会议》（EMNLP），2020 年，第 6769–6781 页。[5] J. Johnson、M. Douze 和 H. J´egou，“十亿规模的 GPU 相似性搜索”，《IEEE 大数据汇刊》，第 7 卷，第 3 期，第 535–547 页，2021 年。[6] G. Graefe，“数据库系统中的排序实现”，《ACM Computing Surveys》，第 38 卷，第 4 期，2006 年。[7] J. Shin、J. Chang 和 F. Nargesian，“ListK：通过列表式提示实现语义 ORDER BY 和限制为 K”，2026 年。[在线]。可访问：https://arxiv.org/abs/2603 17223。S. Dai、W. X. Zhao、J.-r. Xu、Z. Tang、Y. Ye、J. Zhang[8] 和 J.-r. Wen，“揭示 ChatGPT 在推荐系统中的能力”，2023 年。[9] Z. Qin、R. Jagerman、K. Hui、H. Zhuang、J. Wu、J. Yan、L. Shen、T. Liu、J. Liu、D. Metzler、X. Wang 和 M. Bendersky，“大语言模型是有效的文本排序器：采用成对排序提示”，载于《计算语言学协会：NAACL 2024 论文集》。墨西哥城，墨西哥：计算语言学协会，2024 年。[在线]。可访问：https://aclanthology.org/2024.findings-naacl.97/ [10] R. Tang、X. Zhang、X. Ma、J. Lin 和 F. Ture，“发现于中间：置换自洽性改进大语言模型中的列表式排序”，arXiv 预印本 arXiv:2310.07712，2024 年。[11] X. Ma、X. Zhang、R. Pradeep 和 J. Lin，“使用大语言模型进行零样本列表式文档重排序”，arXiv 预印本 arXiv:2302.10584，2023 年。[12] R. Beigel 和 J. Gill，“使用 k-排序器对 n 个对象进行排序”，《IEEE Transactions on Computers》，第 39 卷，第 5 期，第 714–716 页，1990 年。[13] T. Leighton，“并行排序复杂度的紧界”，《IEEE Transactions on Computers》，第 C-34 卷，第 4 期，第 344–354 页，1985 年。[14] I. D. Scherson、S. Sen 和 A. Shamir，“剪切排序：一种真正的二维超大规模集成电路网络排序技术”，载于《并行处理国际会议》（ICPP）论文集，1986 年。[15] Y. Zeng、O. Tendolkar、R. Baartmans、Q. Wu、L. Chen 和 H. Wang，“LLM-RankFusion：缓解基于大语言模型的排序中的内在不一致性”，arXiv 预印本 arXiv:2406.00231，2024 年。[16] W. Sun、L. Yan、X. Ma、S. Wang、P. Ren、Z. Chen、D. Yin 和 Z. Ren，“ChatGPT 擅长搜索吗？考察作为重排序代理的大语言模型”，载于《2023 年自然语言处理实证方法会议论文集》。新加坡：计算语言学协会，2023 年。 [在线]。可访问：https://aclanthology.org/2023.emnlp-main.923/ [17] S. Zhuang、H. Zhuang、B. Koopman 和 G. Zuccon，“使用大语言模型进行高效且高度有效的零样本排序的集合式方法”，载于《第 47 届 ACM SIGIR 信息检索研究与开发国际会议论文集》，2024 年。[18] R. Nogueira 和 K. Cho，“使用 BERT 进行段落重排序”，2019 年。[19] P. Wang、L. Li、L. Chen、Z. Cai、D. Zhu、B. Lin、Y. Cao、L. Kong、Q. Liu、T. Liu 和 Z. Sui，“大语言模型并非公平评估器”，载于《计算语言学协会第 62 届年会论文集》（第 1 卷：长篇论文）。曼谷，泰国：计算语言学协会，2024 年，第 9440–9450 页。[20] R. Pradeep、S. Sharifymoghaddam 和 J. Lin，“Rankzephyr：有效且稳健的零样本列表式重排序易如反掌！”2023 年。 [在线]。可访问：https://arxiv.org/abs/2312 02724。S. Herbold，“Sort Bench：基于大语言模型排序列表能力的基准测试”，[21] 2025 [ ] 2504 08312。在线。可访问：https://arxiv.org/abs/ [22]。T. Joachims，“利用点击数据优化搜索引擎”，载于《ACM SIGKDD 知识发现与数据挖掘国际会议论文集》，2002 年，第 133–142 页。[23] C. J. Burges，“从 RankNet 到 LambdaRank 再到 LambdaMART：概述”，Microsoft Research，技术报告 MSR-TR-2010 82, -。在线。可访问：2010 [ ] https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/

[24] A. Marcus、E. Wu、D. R. Karger、S. Madden 和 R. C. Miller，“由人工驱动的排序与连接”，《VLDB基金会会议录》，第 5 卷，第 1 期，第 13–24 页，2011 年。[25] A. B. Kahn，“大型网络的拓扑排序”，《ACM 通讯》，第 5 卷，第 11 期，第 558–562 页，1962 年。[26] G. Hardy、J. Littlewood 和 G. P´olya，《不等式》，剑桥数学文库丛书。剑桥大学出版社，1952 年。[27] D. J. Pearce 和 P. H. J. Kelly，“用于有向无环图的动态拓扑排序算法”，《ACM实验算法学杂志》，第 11 卷，第 1.7–es 页，2007 年。[28] H. V. Jagadish，“一种用于物化传递闭包的压缩技术”，《ACM数据库系统汇刊》，第 15 卷，第 4 期，第 558–598 页，1990 年。[29] K. Simon，“一种改进的无环有向图传递闭包算法”，《理论计算机科学》，第 58 卷，第 325–346 页，1988 年。[30] V. Satop¨a¨a、J. Albrecht、D. Irwin 和 B. Raghavan，“大海捞针：检测系统行为中的膝点”，载于《国际分布式计算系统会议研讨会论文集》，2011 年。[31] OpenIntro，“OpenIntro统计学：NBA球员身高数据集”，https:// www.openintro.org，2025 年，访问于 2025 年。[32] Y. Hou、J. Li、Z. He、A. Yan、X. Chen 和 J. McAuley，“连接语言与物品以实现检索与推荐”，arXiv 预印本 arXiv:2403.03952，2024 年。[33] D. Wadden、S. Lin、K. Lo、L. L. Wang、M. van Zuylen、A. Cohan 和 H. Hajishirzi，“事实还是虚构：验证科学论断”，载于《2020 年自然语言处理实证方法会议（EMNLP）论文集》。线上：计算语言学协会，2020 年，第 7534–7550 页。[34] F. M. Harper 和 J. A. Konstan，“MovieLens数据集：历史与背景”，《ACM交互式智能系统汇刊》，第 5 卷，第 4 期，第 19:1–19:19 页，2016 年。[35] S. Guo、A. Parameswaran 和 H. Garcia-Molina，“那么谁赢了？利用众包群体动态发现最大值”，载于《ACM SIGMOD数据管理国际会议论文集》，2012 年，第 385–396 页。[36] P. Venetis、H. Garcia-Molina、K. Huang 和 N. Polyzotis，“众包环境中的最大值算法”，载于《国际万维网会议（WWW）论文集》，2012 年。[37] S. B. Davidson、S. Khanna、T. Milo 和 S. Roy，“利用众包群体执行 Top-k 和分组查询”，载于《国际数据库理论会议（ICDT）论文集》，2013 年。[38] X. Chen、P. N. Bennett、K. Collins-Thompson 和 E. Horvitz，“众包环境中的成对排名聚合”，载于《ACM网络搜索与数据挖掘国际会议（WSDM）论文集》，2013 年。[39] U. Feige、P. Raghavan、D. Peleg 和 E. Upfal，“利用噪声信息进行计算”，《SIAM计算杂志》，第 23 卷，第 5 期，第 1001–1018 页，1994 年。[40] M. Braverman 和 E. Mossel，“无需重新采样的含噪排序”，载于《ACM-SIAM离散算法研讨会（SODA）论文集》，2008 年。[41] K. G. Jamieson 和 R. D. Nowak，“利用成对比较进行主动排名”，载于《神经信息处理系统进展（NeurIPS）》，2011 年。

附录 A 符号说明

表 VI 列出了本文使用的符号。

表 VI：常用符号。

符号	含义
n	待排序键（项）的数量
$X = \{x_1, \ldots, x_n\}$	提交给排序算子的输入键列表
c	自然语言排序准则（提示词）
m	列表式批次大小（$m \geq 2$）；拐点扫描尝试 $m \in \mathcal{M}$，然后为每个模型固定一个值
$\mathcal{M}$	候选批次大小（$\{4, \ldots, 14\}$；$\{10, 20, 40, 75\}$，针对亚马逊）
B	一次列表式大语言模型调用中键的子集
$\sigma(B) = (x_1, \ldots\ldots, x_m)$	大语言模型返回的 B 的排列
$a \succ b$	严格偏好：在 c 下，a 排名高于 b
$\mathcal{R}$	保留的有序对 $(a,\ b)$，其中 $a \succ b$
$G = (V,\ E)$	关系图 $V = X$，边 $u \to v$ 当且仅当 $(u,\ v) \in \mathcal{R}$
$H = (X,\ E_d)$	直接边构成的稀疏有向无环图（G 是其传递闭包）
π	关于 X 的最终全序（字典序破同分）

附录 B 其他相关工作

批量比较器。每次调用对 k 个项目进行排序是经典做法：Beigel 和 Gill 将 k-排序器的复杂度限定为 [12]；列排序 [13] 和剪刀排序 [14] 交替执行列阶段和行阶段。TBS 在第 1–3 轮次中复用该网格，但不同于那些完美的比较器 Schedule，它会跳过根据先前答案已经可以推断出的批次，并拒绝矛盾。

众包排序。众包数据库算子同样按比较次数付费，并能容忍噪声 [24], [35]–[38]；其中一些方法通过传递性减少提问次数。TBS 将同一思路应用于列表式大语言模型调用，但有三点不同：每次调用最多产生 $\binom{m}{2}$ 个偏好，Schedule 是批次网格，并且在较大的 n 下图结构必须保持较低开销（由可达性索引处理，§IV-B）。

含噪排序。理论界限研究有无重新采样时随机错误下的比较 [39], [40]，并研究几何假设 [41]下的主动排名。TBS 不假设误差模型：它只保留 G 无环结构，并在经验上测量残余分歧（§C-A）。

附录 C 其他实验结果

A. 量化残余大语言模型不一致性

格式验证（§IV-C）确保回复是其批次的一个排列，而不确保偏好正确。仍存在两项残余不一致性，概述于表 VII。

表 VII：TBS 不一致性（GPT-4o）。 $\varepsilon_{\text{reject}}$、$\eta$、**格式错误：第 2** R_0 **轮审计（*NBA*/流行度/亚马逊：一个实例；*DL19*/*SciFact*/*MovieLens*：前 20 个）。打包调用：完全排序至**u=0**，并使用所有行的第 2 轮；*NBA*/流行度/亚马逊：一个实例；*DL19*/*SciFact*/*MovieLens*：前 20 个。**

指标	*NBA*	*Pop.*	*DL19*	*SciFact*	*ML*	*Amz.*
$\varepsilon_{\text{reject}} \downarrow$	0.067	0.065	0.102	0.113	0.091	0.041
$\eta \downarrow$	0.038	0.063	0.115	0.037	0.178	0.030
打包调用（%）↓	12	11	9	11	5	16
格式错误（%）↓	0.0	0.0	0.4	1.0	0.0	0.0

可检测矛盾。会形成环的边 G 会在合并时被拒绝。在接受 D 条直接边、拒绝 R_{rej} 条直接边的情况下，$\varepsilon_{\text{reject}} = R_{\text{rej}}/(D + R_{\text{rej}})$ 是合并时冲突率，而非大语言模型错误的下界。

无法检测的语义噪声。与 G 一致的边在 c下仍可能是错误的。我们对存储的直接边进行成对式重新询问；如果 C 表示方向与存储边一致的比例，则$\eta = 1 - C$。这两个比率使用不同的分母，不能合并。

表 VII 将格式错误与偏好噪声区分开来。当列表式回复提出一条会使偏好图形成环的边 G时，TBS 会在合并阶段拒绝该边：在各数据集上，$\varepsilon_{\text{reject}}$ 的范围从。041（亚马逊）到 0。113（*SciFact* ）（约占候选直接边的 4–11%）。这一比率表示与已接受结构发生冲突的情况，而不是所有大语言模型错误。一个过早产生的错误边可能阻断之后的正确边，因此即使模型在该批次中回答正确，之后的真实偏好也可能被丢弃。这一影响有助于解释为何停用传递性（TBS-NoTrans）仅在*DL19* （表 V）上提升了质量。

进入 G 的边仍可能存在语义错误。将存储的直接边抽样，并作为新的成对比较重新询问后，分歧率为 η=0.030–0。178：最低的是亚马逊 （提示词中显示价格）和*SciFact*，最高的是*MovieLens*，因为其中的偏好判断更具主观性。

此外，在所有行的第 2 轮中，将打包运行至完成（u=0）时，未解决的打包占大语言模型调用总次数的 5–16%（*MovieLens*最低，亚马逊最高）。

格式错误回复，即输出并非批次的规范排列，比例最高为 1%（*SciFact*），在四个数据集上则为 0%；每条此类回复都会在任何边合并前通过重试修复。

B. *RQ3*：跨模型和任务的稳健性

我们在默认 Schedule（§VI-A5）下，针对各模型选定的 m 对 GPT-4.1、GPT-4o 和 GPT-4o-mini 进行比较（表 VIII；扫描结果见图 2）。Mini 需要更小的 m（例如，人口 τ=0.952 在 m=6 时，而在 m=10时为 0。564）。在全序任务上，GPT-4o 在*NBA*上表现优于 GPT-4.1（0。876/97 次调用，对比 0。817/110 次调用），而 mini 的表现则大幅下降（0。208/306）；人口数据对所有模型都很容易（$\tau \geq 0.93$；GPT-4.1 最高，为 0。972/87）。在相关性/偏好任务上，各模型的质量相近（*MovieLens*0。878–0。879；*SciFact* 0。839–0。871，mini 最佳；*DL19* 0。900–0。921，GPT-4o 最佳），但尽管 mini 通常需要更多调用次数，其美元成本仍最低。

要点。当排序需要从模型参数中回忆事实知识时（例如 *NBA* 身高和人口排序），模型选择最为重要。在其他情况下，不同模型的排序质量相对接近，而成本更多取决于令牌定价，而非调用次数。

C. *RQ4*：批次大小敏感性

接下来，我们考察列表式批次大小 m如何影响不同模型和数据集上的质量与成本，以及如何为 RQ3 实验选择 m。图 2 展示了 GPT-4.1、GPT-4o 的 m 扫描结果，

以及 GPT-4o-mini 在 *NBA*、人口、*DL19* 和 *MovieLens*（contiguousRound1、对角线，然后是 Pack，θ=1.0）上的结果，并记录质量、调用次数、令牌和美元成本；星号标示每个模型所选的 m。

表 VIII：RQ3：所选 m 下各大语言模型的 TBS。每行最优值以粗体标示。

数据集	指标	GPT-4.1	GPT-4o	GPT-4o-mini
NBA	τ ↑	0.817	**0.876**	0.208
	调用次数 ↓	110	**97**	306
	令牌数（百万） ↓	**0.06**	**0.06**	0.10
	成本（美元） ↓	0.33	0.43	**0.04**
人口	τ ↑	**0.972**	0.932	0.952
	调用次数 ↓	**87**	101	202
	令牌数（百万） ↓	**0.04**	0.05	0.08
	成本（美元） ↓	0.18	0.36	**0.03**
DL19	nDCG_Q ↑	0.905	**0.921**	0.900
	调用次数 ↓	2,105	**1,790**	4,522
	令牌数（百万） ↓	2.27	**2.24**	3.40
	成本（美元） ↓	7.67	10.49	**0.85**
SciFact	nDCG_C ↑	0.864	0.839	**0.871**
	调用次数 ↓	**2,032**	2,102	7,392
	令牌数（百万） ↓	**5.91**	6.46	11.48
	成本（美元） ↓	15.60	23.77	**2.42**
MovieLens	nDCG_U ↑	0.878	**0.879**	0.878
	调用次数 ↓	1,893	1,832	**1,784**
	令牌数（百万） ↓	0.90	0.87	**0.86**
	成本（美元） ↓	2.11	2.61	**0.16**

成本与质量。 由于每次列表式调用最多可排序 $\binom{m}{2}$ 个配对，增大 m 可以缩短三轮调度，并减少调用次数、令牌数和美元成本；但当 m 已经较大时，收益会递减。GPT-4.1、GPT-4o 和 GPT-4o-mini 在每个 m下的调用次数几乎相同，因此模型之间的美元成本差异主要由令牌定价决定，而非 TBS 发出的批次数量。然而，质量取决于模型能否对更长的列表生成可靠的全序。在整个测试范围内，GPT-4.1 和 GPT-4o 的质量基本保持平稳：较大的批次可以降低成本，且没有明显的质量损失，因此 §V-C 的拐点/端点规则会选择较大的 m （9–12），以节省调用次数。GPT-4o-mini 在全序和段落排序方面则不具备同样的稳健性。在*NBA*上，mini 的 m=4 已经处于 $\tau\approx$0.21，而 $m\geq$6则跌至接近零；在人口上，质量从较小 m 时的 $\approx$0.97降至 m增大后低于 0.6；在*DL19*上，nDCG_Q 会在 m 超过 5–6 后下降。在这些设置中，较大的 m虽然能减少调用次数，却会导致排序失效，因此选择器会让 mini 使用较小批次（在*NBA*/人口/*DL19*上为m=4/6/5 ）。*MovieLens* 则是例外：对简短电影描述进行偏好排序时，三个模型的表现都保持稳定，选择器为每个模型都选取 m=10。

要点。 当列表变长时质量保持不变（如 GPT-4.1 和 GPT-4o），更大的 m 主要能减少调用次数、令牌数和费用。相比之下，GPT-4o-mini 在事实性全序和段落排序任务上需要较小的 m，因为更长的列表式提示词会削弱较弱比较器的表现。

D. RQ5：执行与批次策略

表 IX 比较了 §V-B 中 GPT-4o 下的四种第1轮成员预设（m=10、递归 R_0、对角线后接 Pack、 θ=1.0）。在各数据集上，连续与随机的质量差异仅为 0。002– 0。024，因此两者之间的选择几乎无关紧要。在*NBA* 和人口数据集上，键只是简单名称，可用于聚类的文本很少；两种相似度预设在 τ 上的表现都不如连续，但调用次数只减少少许。在段落排序和偏好任务中，文本更丰富，但相似度预设仍未占据优势：在*DL19*和*MovieLens*上，连续仍然成本最低或几乎持平；只有*SciFact* 在质量上更偏好词汇相似度。

表 IX：RQ5：第1轮成员关系（GPT-4o， m=10）。每行最佳值以粗体标示。

数据集	指标	连续。随机		相似度。相似度嵌入。
NBA	τ ↑	**0.876**	0.866 0.849	0.832
	调用次数 ↓	97	96 93	**89**
	令牌数（百万） ↓	0.06	0.06 0.06	**0.05**
	成本（美元） ↓	0.43	0.41 0.41	**0.39**
人口	τ ↑	**0.932**	0.908 0.861	0.892
	调用次数 ↓	101	**95** **95**	96
	令牌数（百万） ↓	0.05	0.05 0.05	0.05
	成本（美元） ↓	0.36	0.35 **0.34**	**0.34**
DL19	nDCG_Q ↑	0.921	**0.923** 0.895	0.897
	调用次数 ↓	**1,790**	1,840 1,824	1,806
	令牌数（百万） ↓	**2.24**	2.30 2.27	2.26
	成本（美元） ↓	**10.49**	10.91 10.81	10.69
SciFact	nDCG_C ↑	0.839	0.861 **0.862**	0.838
	调用次数 ↓	2,102	2,110 2,107	**2,080**
	令牌数（百万） ↓	6.46	6.43 6.45	**6.38**
	成本（美元） ↓	23.77	23.73 23.71	**23.48**
MovieLens	nDCG_U ↑	**0.879**	0.877 0.873	0.873
	调用次数 ↓	**1,832**	1,855 1,854	1,841
	令牌数（百万） ↓	**0.87**	0.88 0.88	0.88
	成本（美元） ↓	**2.61**	2.66 2.66	2.64

要点。 第1轮的成员划分变化对质量的影响大于调用次数，而在大多数数据集上，连续或随机都是稳定的选择。在 *NBA*/人口上，连续方式表现最佳（τ=0.876/0.932），相似度方式则落后 0.027–0.071，且调用次数节省很少，这与裸名称键只能提供较弱聚类信号的情况一致。在 *DL19*/*MovieLens*上，连续方式的成本最低或并列最低（随机方式在 *DL19* 上以 0.923 并列）；相似度方式在 *DL19*上落后 $\approx$0.025 ，在 *MovieLens*上则始终相差不超过 0.006。*SciFact* 是例外：词汇相似度（0.862）与随机方式相当，并优于连续方式（0.839），这可能是因为共享的科学术语将相关摘要置于同一第1轮列中，使列表式比较更容易；而嵌入聚类体现的是主题，不一定与主张级支持或反驳相一致；嵌入相似度的总体成本最低，但并未提升质量。

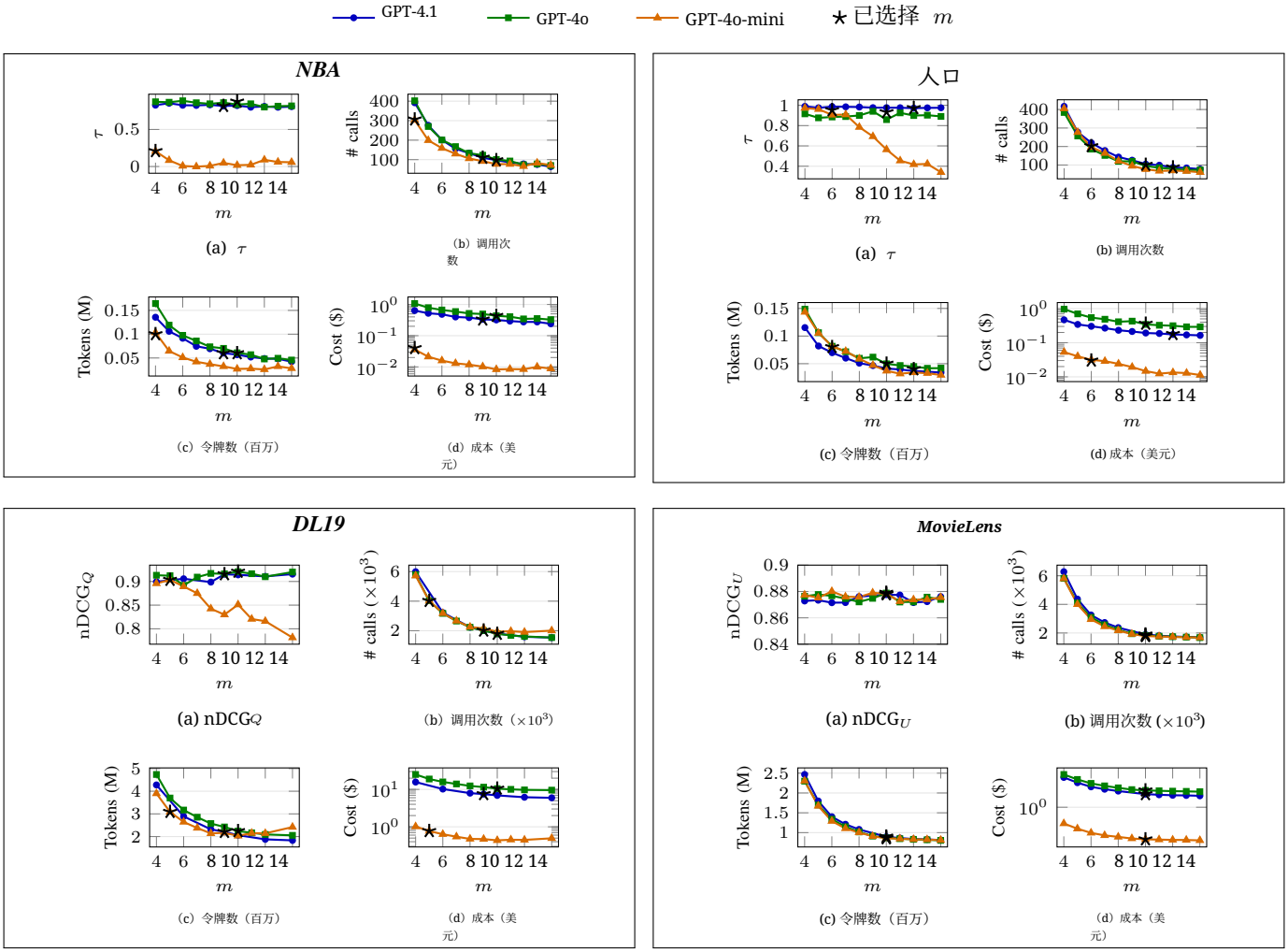


图2：RQ4：批大小 m 与质量、调用次数、令牌和成本（美元）的关系。每个模型一条线；星号表示选定的 m （相同 m 下星号与线之间的差距为运行变化）。连续第一轮，随后为对角线和 Pack， θ=1.0；除 *DL19*第二轮窗口外均为递归 R_0 。