arXiv:2602.14089v1 [cs.DB] 15 Feb 2026

TabTracer：大语言模型的复杂表格推理蒙特卡洛树搜索

Zhizhao Luo[1,2], Zhaojing Luo[2], Meihui Zhang[2]，*，Rui Mao[3]1北京理工大学珠海校区，
[2]北京理工大学，[3]深圳大学 {zzluo,zjluo,meihui_zhang}@bit.edu.cn,mao@szu.edu.cn

摘要

大语言模型（LLM）已成为自然语言表格推理的强大工具，其方法主要分为两类。基于提示的方法依赖纯语言推理，或一次性生成程序，且不进行步骤级验证。基于智能体的方法在闭环中使用工具，但验证往往局限于局部，回溯能力有限，因此错误容易传播，成本也随之增加。此外，这类方法依赖链式或束搜索式轨迹，而这些轨迹通常存在组合冗余，导致令牌成本高昂。本文提出 TabTracer，这是一种智能体框架，可协调针对中间表格状态的多步工具调用，并通过显式状态跟踪实现验证和回滚。首先，它通过类型化操作以及轻量级数值和格式检查来强制执行步骤级验证，从而提供可靠的奖励并抑制幻觉。其次，基于执行反馈的蒙特卡洛树搜索维护由候选表格状态构成的搜索树，并利用反向传播的反思分数指导 UCB1 选择，通过版本化快照实现回滚。第三，它通过预算感知剪枝、去重和带单调性门控的状态哈希来减少冗余，从而降低令牌成本。在 TabFact 、 WikiTQ 和 CRT 数据集上的全面评估表明，TabTracer 的准确性比最先进的基线方法最高提升 6.7%，同时将令牌消耗降低了 59%–84%。

1 引言

结构化表格是组织数据的普遍媒介 [3,14] ，广泛应用于医疗保健分析 [7], 、金融风险管理 [2], 和科学发现 [5]等领域。然而，其显式结构往往带来僵化的约束，与自然语言的灵活性不相匹配，因此难以将含糊的用户意图映射为精确的表格操作，以进行组合推理 [13]。为应对这些复杂性，表格推理研究如何根据表格证据 [30]回答自然语言问题。

早期的表格推理方法主要依赖可执行的语义解析或程序归纳 [30, 31]。尽管这些方法具有可解释性，却需要大量符号工程和搜索，且在弱监督、非规范单元格值以及模式/领域偏移的情况下表现脆弱 [20, 54]。为减轻工程负担并提升鲁棒性，研究者引入了面向表格的预训练 [18, 49] ，通过学习更强的文本-表格表示，实现无需显式逻辑形式的端到端推理。然而，这一研究方向仍依赖特定任务微调，因此跨任务和跨领域的泛化能力依然受限。相比之下，大语言模型（LLMs） [19, 46] 具有更强的上下文内泛化能力和更广泛的跨模式迁移能力。

基于大语言模型的表格推理将大语言模型视为核心推理引擎 [36] ，并依靠提示来驱动表格理解，推理和作答。早期的基于大语言模型的方法大多基于提示 [10, 42, 57]。这类方法一次性输入表格和问题，或采用固定的多阶段提示，其中步骤顺序预先设定，各阶段的输出单向传递，且不会根据中间执行结果改变后续决策。由于没有外部执行信号来确认中间假设（例如所选列或筛选出的行）是否正确，一旦模型误读模式或保留了错误的子集，后续推理便会建立在这一错误之上。图1中的思维链 [45]（星形）展示了仅依靠语言判断如何引入计数或统计错误，并使错误沿链式搜索传播。这些局限促使基于智能体的方法 [15, 48, 55],应运而生：通过让大语言模型（或多个大语言模型）选择操作、调用工具，并根据执行反馈更新决策，从而形成闭环。然而，反馈往往局限于局部，系统性回溯能力有限，而多分支推理会增加令牌成本。图1中，ReAcTable [53] （月亮-1）未能触发工具使用，并在代码生成前陷入停滞；Chain-of-Table [44]（月亮-2）累积了过程错误，却未能及时回滚，将月份误读为“日期”，也未能按月份汇总；Table-Critic [50] （月亮-3）的结果虽接近正确，却在表达形式上不匹配，且多分支对话带来了高昂的令牌成本。总体而言，现有方法在准确性和效率方面仍面临明显挑战。

挑战 1： 现有方法缺乏步骤级验证和可靠的奖励，因此中间错误往往未得到检查便不断传播。将表格线性化供大语言模型使用时，行列对齐以及模式、类型或单位约束等二维结构线索会被削弱 [25, 29], ，使语义错误和表格幻觉更难被发现。大语言模型对计数和聚合任务中的数值精度也不敏感，因而会产生数值幻觉 [8, 23], 并在后续步骤中进一步累积。我们需要机器可检查的步骤级验证，以及能够体现意图、基于执行的奖励；轻量级数值和格式检查 [6] 可使验证切实可行，并确保推理以证据为依据 [34, 44, 53]。这很困难，因为表格算子和数据格式差异很大，难以定义统一的前置条件和后置条件 [26, 37]。此外，许多工具接口采用自由格式或解析能力有限，因此难以构建可靠的奖励，也使中间验证不可靠。

挑战 2： 缺乏有效的回溯，使得早期错误难以纠正。大多数方法采用线性或浅层分支流程，因此一旦问题或表格在早期被误读，后续步骤就只能在当前路径上进行局部修补 [16]。它们很少回退关键决策或探索替代方案，导致错误根深蒂固并不断放大。更好的方法是采用可回溯推理机制：当执行证据与当前假设相冲突时，能够返回关键决策点，重置或替换子路径，

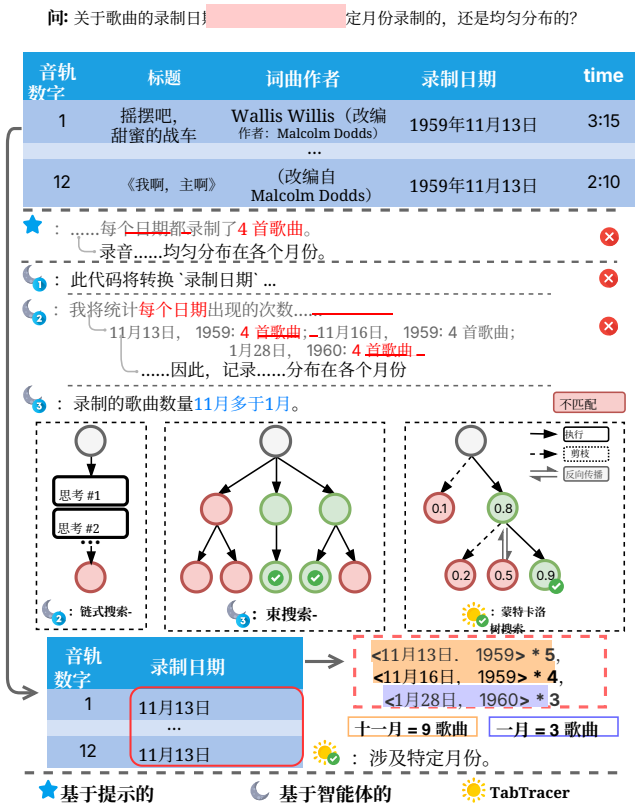


图1：基于提示的和基于智能体的输出未能完成聚合，而 TabTracer（我们的方法）通过切分表格，按日期统计歌曲数量，并按月份汇总（十一月=9 与一月=3）。

并根据证据在候选分支中进行选择 [35]。然而，这颇具挑战性。执行反馈通常具有局部性且噪声较大，因此难以形成稳定的价值估计或置信度分数来支持可靠回溯；系统还必须在继续当前路径与回退以探索替代方案之间取得平衡，避免出现振荡或过度回溯。

挑战 3： 在令牌与延迟约束下，目前尚无能够抑制冗余分支、同时保持稳定进展的预算感知搜索方法。许多多分支或多智能体方法采用链式搜索或束搜索拓扑结构 [11, 41, 47]。如图1所示，链式推理无法回溯，因此早期的错误会导致整条路径崩溃。图1中的束搜索会保留多个分支，但其中许多分支几乎重复，从而造成冗余扩展，并增加令牌与执行成本 [60]。更稳健的策略是结合基于执行的优先级排序、剪枝和去重，以确保在受限预算内实现有效搜索 [9, 17]。然而，这种方法在实际实施中面临诸多困难。一个主要障碍源于执行反馈本身具有噪声且局部性强。这些局限可能导致剪枝启发式方法无意中丢弃有效的推理路径，却保留误导性的路径。此外，防止搜索重复访问冗余的表格状态也是一项重大挑战。例如，交替执行表格缩减与扩展操作可能导致搜索过程退化为循环 [56]。虽然从理论上说，将搜索空间构造为有向无环图（DAG）是提高效率的理想做法，但在实际实现中仍难以确保其无环性。

本文提出了 TabTracer，这是一种智能体框架，可在中间表格状态上运行执行与反思循环。系统首先对表格和问题进行轻量级分析，提取列名和类型、缺失值模式、表格规模及问题类型，以初始化初始状态并约束行动空间。首先，为了确保步骤级验证并提供以执行为依据的奖励，TabTracer 使用类型化表格运算符执行列选择、行筛选和计算；每项操作都会生成一个带版本的中间表格，并附有可由机器检查的前置与后置检查。这一机制确保推理过程始终以可验证的证据为依据，从而有效抑制数值幻觉，并防止错误在步骤间传播。其次，为了支持可靠回溯，基于执行反馈的蒙特卡洛树搜索（MCTS） [4] 维护一棵由候选表格状态构成的树，并反向传播反思分数以更新节点值，从而进行 UCB1 选择；同时依靠版本化快照，在证据发生冲突时支持回滚和子路径替换。第三，为了控制冗余和令牌成本，我们通过剪枝和去重实施固定的令牌预算。状态哈希与复用可抑制近乎重复的扩展；单调性门控则仅在语义状态确实发生变化时才提交节点，从而形成类有向无环图式进展，并减少预算内的冗余执行。如图1所示，TabTracer 首先截取一个聚焦的子表（例如曲目／录制日期），然后通过经状态验证的工具执行进行定向聚合，得出最终答案。由此生成的搜索树更加紧凑，以更少的扩展次数得出答案，并降低令牌用量。

总而言之，我们的主要贡献如下：

- 我们提出了 TabTracer，这是一种智能体框架，通过使用类型化操作符和步骤级检查，生成可靠的执行奖励并减少幻觉。
- 我们开发了执行反馈蒙特卡洛树搜索框架，该框架维护候选表格状态，并通过带版本的快照支持基于证据的回溯，从而纠正早期错误，而非修补当前路径。
- 我们提出了预算感知剪枝和去重方法，结合状态哈希、单调性门控和失败记忆，在固定预算下抑制近乎重复的扩展并减少令牌消耗。
- 我们在 Tab-Fact、WikiTQ 和 CRT 数据集上进行了全面评估，结果表明，TabTracer 的准确性优于最先进的基线方法，最高提升 6.7%，同时令牌消耗减少 59–84%。

本文其余部分安排如下：第 2 节回顾相关工作，第 3 节详细阐述 TabTracer 方法，第 4 节介绍实验设置与结果，第 5 节总结全文。

2 相关工作 2.1 基于提示的推理方法

基于提示的方法将表格和问题组织成单轮提示或固定的两阶段提示。大语言模型不会根据外部执行反馈进行多步决策，因此其推理依赖于仅基于语言的推理或单次程序生成。我们将这类方法分为纯语言自推理和固定流程程序两类。

纯语言自推理。这类方法不使用可执行程序和外部反馈，而是通过提示结构和表征来引导推理。Tab-CoT [61]通过结构化的 Markdown 推理表格对齐行和列。两阶段自增强提示会在推理前提取结构线索，从而提升鲁棒性 [38]。简单的表格序列化只需少量上下文示例，即可提升表格推理性能 [10]。FLEXTAF [52] 通过多格式选择和投票来改善对齐效果。这类方法主要改进提示结构、表征和稳定性，但仍缺少工具级验证和回滚，因此难以纠正早期偏差。在本研究中，TabTracer 将语言推理建立在可执行工具之上，并引入步骤级可验证反馈和基于反思的奖励，从而减少格式正确但语义错误的步骤不断累积。

固定流水线程序。这类方法只生成一次程序或提取指令，执行结果不会反馈到后续决策中，因此形成单向流水线。Binder [12] 将大语言模型 API 嵌入混合程序，但执行流程仍是固定的。TabSQLify [27] 采用 SQL 到子表再到答案的工作流，而 H-STAR [1] 则采用由固定提取和路由推理组成的两阶段方法。LOTUS [33] 引入了语义算子和查询优化。TabTrim [16] 通过黄金轨迹监督剪枝处理并行子表配置。这些方法高度依赖提取或程序生成的语义正确性，一旦子表或程序出错，就很难恢复。相比之下，TabTracer 以信息增益蒙特卡洛树搜索取代单向流水线，并通过带哈希去重的带版本表格状态实现回溯和复用。

2.2 基于智能体的推理系统

基于智能体的方法将大语言模型视为控制器，使其在闭环中进行多步决策，并利用工具、检索或执行反馈来更新后续行动。我们将其分为工具使用型智能体和多智能体协作。

工具使用型智能体。这类方法侧重于通过反复调用工具并利用反馈，提高执行的可行性和可控性。Chain-of-Table [44] 通过迭代操作演化表格。ReAcTable [53] 将 ReAct 引入表格问答，并利用执行反馈，在多条轨迹之间进行投票。AixelAsk [51] 将推理表述为结合分解与检索的有向无环图优化。TableMind [21] 通过监督微调和 RAPO 训练工具使用策略。尽管有反馈，验证往往仍停留在语法或局部执行层面，语义检查和回溯能力有限。在本研究中，TabTracer 增加了步骤级的机器可检验验证与反思奖励，并将工具执行与蒙特卡洛树搜索式回溯相结合，以纠正早期错误。

多智能体协作。这些方法着重于多个智能体之间的协同，这些智能体共享上下文、进行评述或迭代优化结果。AutoTQA [59] 采用多角色有限状态机调度来处理多表问答。MACT [58] 引入在线规划。Table-Critic [50] 依靠多角色评述和自演化模板树。TALON [22] 采用面向长表格的探索-检索-回答流程。然而，验证仍主要基于语言；多智能体对话会增加令牌成本和延迟，而且系统通常只在当前路径内修正，而非进行系统性回溯。我们的方法通过状态复用进行预算感知搜索，避免了多智能体带来的开销，并通过将已知不良状态-动作对列入黑名单来抑制冗余分支，从而以较低成本维持稳定的探索。

3 方法

为应对表格推理中的三项挑战，即逐步验证能力弱且奖励不可靠而导致幻觉、缺乏有效回溯，以及搜索成本冗余，TabTracer 是一种智能体框架，将基于大语言模型的规划与语义表格状态上的环境交互相结合，同时显式管理这些状态，以支持验证与恢复。如图2所示，该框架首先利用**推理层** 运行预算感知的蒙特卡洛树搜索，以管理分支、支持回溯并抑制近乎重复的路径。随后，通过带有前置/后置条件检查的类型化数据框工具，由**执行层** 执行所选行动。最后，**存储层** 保留带版本的表格状态，从而支持回滚和复用。

形式化地，我们将分层智能体表示为：

$$\mathcal{G} = (\mathcal{L}_{\text{reason}}, \mathcal{L}_{\text{exec}}, \mathcal{L}_{\text{store}}), \tag{1}$$

其中，$\mathrm{L}_{\text{reason}}$ 承载规划器，L_{exec} 提供确定性数据框工具（列选择、行筛选和基于代码的计算），$\mathrm{L}_{\text{store}}$ 则提供哈希索引回滚缓存，用于存储带版本的表快照，以便复用和回滚。后续小节将逐一阐述各层，然后分析它们共同提供的保证和运行时优化。

3.1 推理层

该层首先进行表格感知的数据分析，随后运行预算受限的蒙特卡洛树搜索控制器和反思循环，以决定下一步发出哪个工具调用。因此，我们首先介绍分析例程及其所形成的状态抽象，然后在第 3.1.2 节详述信息引导的蒙特卡洛树搜索更新，最后说明由此产生的分数如何为后续各层所依赖的收敛保证提供依据。

3.1.1 数据分析。 图2左侧模块展示了数据分析。我们将完整表格加载到数据框中，记录行列计数，检测特殊行/列或缺失值（例如平均值 列是否完整），并生成 Markdown 预览，作为规划器的上下文记忆。与此同时，我们会改写并分类问题（数值型、比较型、查找型等），分析器则会提出一个初始工具骨架。这些元数据构成根快照——其中记录了表格 URI、语义哈希、规划器/工具轨迹以及最新的反思——足以支持下游规划。

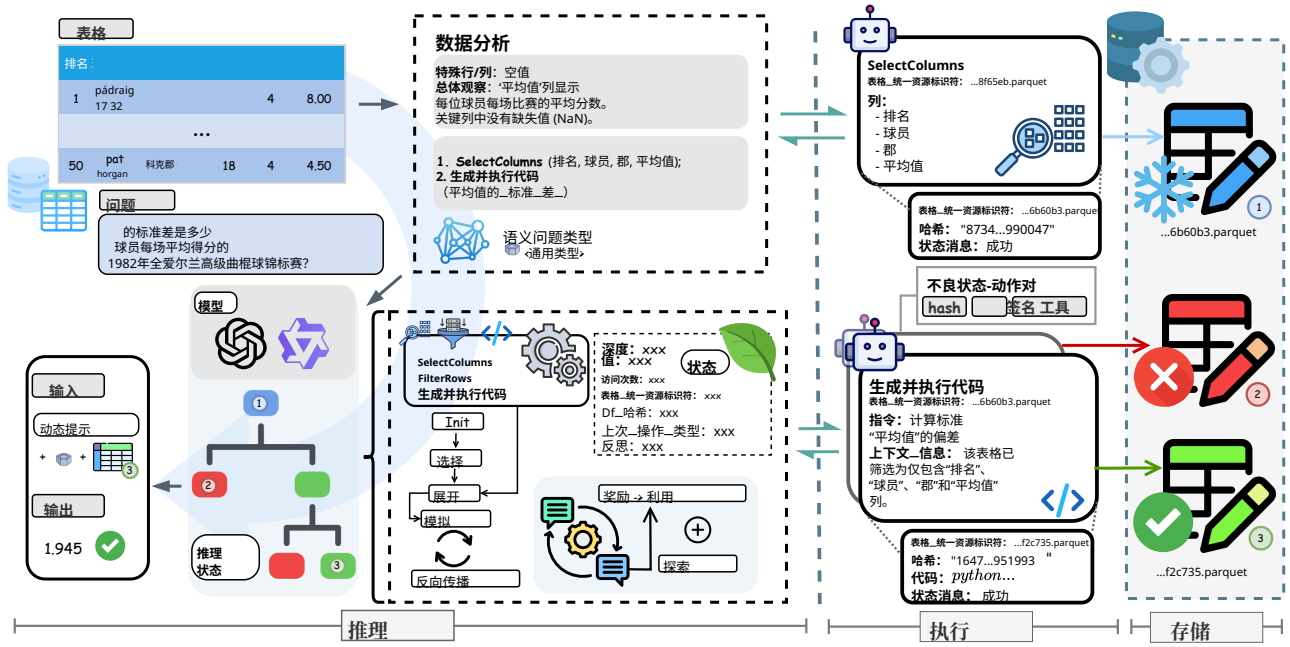


图2：推理层包含规划和反思，执行层调用原子数据框工具，版本化存储层则保留快照，以便回退和重试。

此处的快照 指工具执行后持久化的表格状态（由其 URI 和哈希标识），而轨迹 则记录生成该状态的推理消息。形式上，我们将 TabTracer 视为一个马尔可夫决策过程 M =S, A, T, R π，其状态空间由这些持久化快照及其语义轨迹构成：

$$\mathcal{S} = \{\langle u, \chi, \mu, r\rangle : u \in \mathcal{U},\ \chi \in \mathcal{H},\ \mu \in \mathcal{M}_{\text{msg}},\ r \in \mathcal{R}_{\text{refl}}\}. \quad (2)$$

此处 u 是 Parquet 快照的统一资源标识符， χ 是由确定性工具返回的语义哈希（根据数据框内容生成的确定性指纹），μ 存储生成该状态的有序规划器/工具消息（作为此前记为 C 的语义上下文）， r是最近一次对执行反馈进行总结的反思。有限动作集为 A ={FilterRows, SelectColumns, GenExeCode}。其中，SelectColumns 用于缩减列宽，FilterRows 使用受保护的谓词筛除行，GenExeCode 则执行聚合或多步计算。详情见第3.2节。动作对应确定性工具。状态转移 T 会加载由 u_t 指定的快照，应用所请求的工具，并将后继状态及其哈希写回存储。奖励 R 是归一化反思分数。由于确定性工具执行意味着：这一紧凑记录是马尔可夫充分的

$$P(s_{t+1}, r_t \mid s_{0:t}, a_{0:t}) = P(s_{t+1}, r_t \mid s_t, a_t), \quad (3)$$

因此，语义哈希可作为证书，证明两个节点编码的是同一个数据框；而消息/反思轨迹则保留后续提示所需的语义上下文。TabTracer 不维护完整的哈希账本，而只存储当前快照和无效状态–动作签名缓存，从而使规划器保持可观测，同时避免内存膨胀。这种状态抽象支持下游规划，并使存储的快照足以满足后续工具使用的需求。

3.1.2 信息引导的蒙特卡洛树搜索周期。 我们在表状态上定义搜索树。每个节点都存储一个快照及其推理轨迹，每条边对应一个转换表格状态的单一动作，通常通过工具调用实现。顺序执行而不进行回溯，会形成脆弱的推理链，早期错误会污染下游计算。TabTracer 采用信息引导的蒙特卡洛树搜索来维护多个候选路径，并从次优决策中恢复。根据上述状态视图，规划器在每次模拟中重复以下四个阶段：**(i) 选择** 依据偏向探索的分数遍历现有搜索树。**(ii)扩展** 在确定性保护条件下执行最有前景的归一化工具提案。**(iii) 反思** 向反思算子查询有关新建节点的稠密反馈；**(iv) 回传** 将标量奖励和已解决状态向根节点传播。完整流程在算法1中形式化定义。为利用结构化状态，规划器在每次模拟中执行单个工具，通过反思获取稠密奖励，并依据公式（6）中的分数为节点排序。

虽然反向传播使用标量奖励 r_t，但推演层面的目标会汇总多个执行信号。令 ϕ_t =(ϕ_t^{row}, ϕ_t^{col}, ϕ_t^{rel}, ϕ_t^{sol}) 收集从执行元数据中导出的行/列缩减、工具相关性和解答置信度启发式。推演目标如下：

$$\mathcal{J}(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T-1} \gamma^t\, f_{\text{eval}}(\phi_t)\right], \quad (4)$$

其中， f_{eval} 表示整合这些信号的反思评分器。反思模块通过结构化提示接收 ϕ_t 的所有组成部分（包括公式 (8) 中的行列计数、工具元数据和问题上下文），从而实现依赖上下文的整合。由此得到的反思分数 r_t 表示一种隐式

算法1 TabTracer 单步蒙特卡洛树搜索

要求： 初始状态 s_0、模拟预算 B、黑名单 K

1：初始化根节点 n_0，其为 s_0
2: **对于** $t = 1$ 到 B **执行**
3: $n \leftarrow$选择叶节点(n_0) {UCB1 选择并进行约束}
4: $(child,\ a,\ s',\ \rho_{n,\ a}) \leftarrow$ExpandNode($n$, K) {Alg. 2}
5: **如果** $child = \perp$，**则**
6: **继续**
7: **结束if**
8: $(score,\ solved,\ evidence) \leftarrow$反思($n.state,\ a,\ s'$) {Alg. 3}
9: 反向传播($child,\ score,\ solved,\ \rho_{n,\ a},\ evidence$) {update 值/访问次数和已解决标志}
10: **若** $solved$ **则**
11: **中断**
12: **结束若**
13: **结束循环**
14: **返回** BestSolution(n_0)

评估 $r_t \approx f_{\text{eval}}(\phi_t)$，其中大语言模型会根据问题类型和数据特征自适应地为各项特征赋权。

对于回退排序，我们还维护针对重复状态或退化状态的惩罚指标：

$$\Phi_{\text{pen}}(\tau) = \sum_{t=0}^{T-1} \gamma^t \Big(\lambda_{\text{dup}} \mathbf{1}_{\text{dup}}(s_{t+1}) + \lambda_{\text{empty}} \mathbf{1}_{\text{empty}}(s_{t+1}) \Big), \tag{5}$$

其中 $\mathbf{1}_{\text{dup}}$ 检测哈希重复状态（$\delta = 0$ 见式(7)），$\mathbf{1}_{\text{empty}}$ 标记退化表。这些惩罚被惰性求值，并由后文所述的备用评分器使用。

实际上，TabTracer使规划循环在分析上保持轻量，同时将执行感知启发式方法恰好嵌入其发挥作用之处，即推演目标内部，从而使调度器在理论上保持简洁，并在实践中具有数据效率。

算法1概括了单步蒙特卡洛树搜索循环。第2–3行初始化根节点状态，并在模拟预算上迭代。每次迭代选择一个叶节点（第4行），用一次工具操作对其进行扩展（第5行），对结果进行反思（第6行），并反向传播分数（第7行）。若找到解，循环提前终止（第8–9行）；搜索持续至预算耗尽，并返回最佳解（第11行）。

选择阶段。 在图2中，选择阶段会从当前候选项中挑选 UCB1 分数最高的叶节点。对于此查询，该叶节点通常会保留 *平均值* 列，以便后续操作计算标准差。选择过程采用轻量级 UCB1 分数，在保留 UCB1 简洁性的同时纳入执行感知防护机制：

$$\text{UCB1}(c) = c.value + C\sqrt{\frac{\log(n.visits + 1)}{c.visits + \epsilon}}, \tag{6}$$

其中，c 是 n（父节点）的子节点，C 是探索常数，ϵ 用于避免除以零。访问次数为零的子节点会获得近乎无限的优先级，因此搜索会先快速探索尚未访问的分支，再细化高值分支。选择阶段会从根节点开始迭代向下遍历，在符合条件的节点中选择 UCB1 分数最高的子节点，并在遇到第一个未访问节点时停止。完成评分后，TabTracer 会应用执行感知防护机制——基于哈希的重复检测、空表回滚以及单调性门控——以确保前沿上只保留真正具有信息价值的状态。

扩展阶段。 对于每个选中的叶节点——即代表当前表快照的叶节点——扩展过程（算法 2）会使用大语言模型提出一组有序的候选工具操作，应用确定性有效性过滤器（列存在性、参数有效性、缓存的不良状态—操作签名、重复/无效操作以及工具防护机制），并执行第一个有效操作以创建子节点。我们定义语义状态变化指标。在图2中，规划器会提出多个候选操作，然后过滤掉无效或质量较低的操作，再执行第一个有效的工具调用，例如使用 SelectColumns 保留 {rank, player, county, average}。执行后，生成的子表将保存为新的快照，并成为下一阶段的子节点。

$$\delta(D, D') = \begin{cases} 1 & \text{if } \chi(D') \neq \chi(D) \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

其中，χ 表示内容哈希。随后，缩减代理指标会追踪数据缩减：

$$\rho(D, a) = \delta(D, D') \cdot \phi_{\text{rows}}(D, D') \cdot \phi_{\text{cols}}(D, D'), \tag{8}$$

其中，$\phi_{\text{rows}}(D,\ D')$ = 行(D')/行(D)，而 $\phi_{\text{cols}}(D,\ D')$ = 列(D')/列(D)表示相对维度变化。实现中将 δ 用作硬单调性门控：仅当 $\delta = 1$时才会创建子节点，从而确保每次扩展都会改变表格状态。维度比会作为元数据进行跟踪，并提供给反思模块用于质量评估。

由于执行器仅在语义哈希发生变化时才会实例化子节点，因此每次被接受的扩展都对应着已更新的表格状态。这种单调性有助于规划器避免重复访问。如果某个操作没有改变表格（$\delta = 0$，因此 $\rho = 0$），就会立即将其丢弃。我们仍会设置固定的深度和模拟预算作为硬性保障，但收缩信号能在执行时直接衡量证据的保留程度，从而据此对扩展进行排名。

算法 2 详细说明了这一步。首先，它要求大语言模型为当前状态提出一组数量较少且有序的候选动作（第 2 行）。每个候选动作都会被规范化为具体的工具调用（第 4 行），然后经过有效性检查和缓存的无效签名筛选（第 5–8 行）；通过这些检查的第一个候选动作将被执行，以生成新的快照和子节点（第 9–11 行）。未通过验证的候选动作将被跳过。

候选动作的选择遵循有序提议，并设有明确的有效性门槛：

$$\mathcal{A}_{\text{cand}} = [a_1, \ldots, a_k] \sim p_\theta(\cdot \mid s_t), \tag{9}$$

$$\mathcal{A}_{\text{valid}}(s_t) = \{a \in \mathcal{A}_{\text{cand}} : \textsc{Valid}(a, s_t)\}, \tag{10}$$

$$i^* = \min\{i \in \{1, \ldots, k\} : a_i \in \mathcal{A}_{\text{valid}}(s_t)\}, \quad a_t = a_{i^*}. \tag{11}$$

其中，大语言模型的提议顺序提供了隐式优先级，而确定性过滤器则确保操作可执行。这实现了与实际实现一致的受约束首个有效项选择规则。这里，$p_\theta(\cdot \mid s_t)$ 表示以状态 s_t 为条件的大语言模型诱导提议分布，而 Valid($a,\ s_t$) 表示确定性的可执行性检查，包括列存在性、参数有效性、不良配对过滤、重复/无效项过滤以及工具防护机制。

算法 2 ExpandNode

要求： 节点 n，黑名单 K，候选预算 k

1: $\mathrm{A}_{\text{cand}} \leftarrow$ LLMPropose($n.state, k$) {有序候选项}
2: **对于** a 在 A_{cand} **执行**
3: ($tool$, $args$) ← NormalizeAndCoerce(a)
4: **如果** $tool = \varnothing$ **那么**
5: **继续**
6: **结束如果**
7: **如果** ColumnOrArgInvalid(tool, args, n)**那么**
8: **继续**
9: **结束如果**
10: **如果** $(n.hash, tool, args) \in \mathcal{K}$**则**
11: **继续**
12: **结束如果**
13: **如果** DuplicateOrIneffective(n, tool, args)**则**
14: **继续**
15: **结束如果**
16: **如果** ViolatesToolGuards(tool, args, n)**则**
17: **继续**
18: **结束如果**
19: $s' \leftarrow \mathrm{T}$ ($n.state$, $tool$, $args$)
20: $child \leftarrow$ NewNode(s', 元数据)
21: 将 $child$ 追加到 $n.children$
22: **返回**($child$, $tool$, s', 元数据)
23: **结束循环**
24: **返回**($\perp$, $\varnothing$, $\varnothing$, 1)

反思阶段。对于板球查询，反思会验证新快照仍包含<日期 id='3'>平均值 </日期>列，且适用于计算其标准差，然后为其评分。丢失必需列或产生不可用表格的候选项会被记录为不良状态–动作对。反思评估器会使用（s, a, s'）以及行/列差异和预览，并仅返回归一化标量分数 $r \in [0, 1]$、布尔型已解决标志，以及存储在各节点中的自由形式评述。这种稠密奖励会立即反向传播，因此在线蒙特卡洛树搜索循环会准确维护算法 3 中描述的统计数据。执行元数据（哈希值、行数、工具名称）会与反思一同缓存，使备用评分器之后能够获取更丰富的信号，同时不影响在线传播。因此，当树需要给出最终答案时，我们会根据缓存的元数据和反思，组装一个延迟计算的特征向量：对前沿节点进行排名时，系统会构造一个特征向量：

$$z_t = [\, r_t,\ \delta_t,\ \phi_t^{\text{row}},\ \phi_t^{\text{col}},\ \mathbf{1}_{\text{solve}} \,]^\top, \tag{12}$$

由归一化反思分数 r_t、状态变化指标 δ_t、维度变化和解决状态组成。这些组件通过结构化提示（包括工具输出和元数据）提供给反思模块，从而实现综合质量评估。通过反向传播（公式 14）得到的累积节点值 V_n代表这一综合分数。随后，备用选择使用：

$$n^* = \arg\max_{n \in \mathcal{N}_{\text{frontier}}} V_n - \beta_{\text{pen}} \cdot \mathbf{1}_{\text{degenerate}}(n), \tag{13}$$

其中，$\mathrm{N}_{\text{frontier}}$ 表示当前前沿节点集，惩罚项用于抑制空状态或重复状态。

算法 3反思

要求： 状态 s，动作 a，后继 s'

1: $prompt \leftarrow$ 组合提示词(s, a, s')
2: $resp \leftarrow$ ReflectOperator($prompt$) {结构化输出：反思，分数，标志}
3: **如果** $resp = \varnothing$ **则**
4: **返回** (0, 假,“反思不可用。”)
5: **结束如果**
6: $score \leftarrow \text{clip}(resp.score/10, 0, 1)$
7: $solved \leftarrow resp.found_solution$
8: **返回** $(score, solved, resp.reflections)$

算法 3 明确展示了这一过程：它根据 (s, a, s') 构建结构化提示，调用反思算子，并返回用于反向传播的分数、已解决标志和评述。

因此，反思仅产生一个归一化分数、一个布尔标志，以及附加到每个节点的评述。退化检查（哈希重复剪枝、空表回滚、工具防护机制）在反思之前运行，而收缩或解置信度线索等更丰富的启发式信息，则在回退期间从缓存元数据中读取。

备份阶段。回传将反思分数和已解决标志向上传播，以更新节点值和访问次数，供 UCB1 选择使用并实现提前终止。经反向传播的分数会沿根路径更新节点值（反思分数的增量均值）和访问次数，从而提高其快照保留了查询所需证据的分支的 UCB1 分数（例如，平均值列）。这使得后续选择更有可能重新访问这些分支，下一次扩展便可在这些分支上调用 GenExeCode 来计算最终统计量，并开启新一轮循环迭代。反向传播过程使用增量均值向上传播奖励：从被扩展的节点开始，我们使用式 (14) 中的增量均值更新其访问次数和值，然后沿父链重复此更新，直至到达根节点。如果反思将该节点标记为已解决，我们便向上传播已解决标志，这可以触发提前终止。

$$V_n \leftarrow V_n + \frac{R_{\text{child}} - V_n}{N_n}, \tag{14}$$

其中 R_{child} 是扩展子节点返回的归一化反思奖励，N_n 是节点 n更新后的访问次数。

每个节点还会存储轻量级执行元数据（最近工具类型、输出行、哈希值），备用评分器随后在对叶节点排名时会使用这些元数据：增强启发式将这些缓存计数器与最终反思相结合，以恢复收缩、相关性和解置信度项，尽管在线反向传播仅保留（$value, visits$）。搜索深度受显式深度上限和收缩率的约束，

$$t \ \le\ \min\Big(d_{\max},\ \log_{1/\eta}\big(\min(R_0, C_0)\big)\Big), \tag{15}$$

且可接受轨迹是那些通过步骤级工具防护的轨迹，

$$\mathbf{1}_{\text{adm}}(\tau) = \prod_{t=0}^{T-1} \mathbf{1}\big[\textsc{Guards}(s_t, a_t) = \text{True}\big], \tag{16}$$

因此，无效操作会立即被过滤，而哈希门控可防止重复访问相同的表格状态。

3.1.3 答案选择与终止。 推理循环会在以下任一情况发生时终止：根节点被标记为已解决（某个子节点已找到证书），或模拟预算耗尽。在这两种情况下，公式 (12) 都会惰性计算一个特征向量，根据累积的反思分数、收缩和退化惩罚对前沿节点进行排名，同时抑制已列入黑名单的状态-动作签名。繁重的工作——存储哈希值、元数据和惩罚权重——由存储层处理（第 3.3 节）；推理层只需查询此缓存以获取 $(z_t,\ s_t^{\text{fb}})$，并选择排名最高的节点来提取答案。

3.2 执行层

该层对应图2中的中间模块：规划器选择的每个操作，都会落实为三种带类型的数据框操作符之一，其输出为可由机器检查的元组（table_uri，hash，rows，cols，preview，status）。目标是将抽象指令转化为确定性的表格操作，同时持续返回用于生成收缩信号和反思提示的元数据。我们的工具设计着眼于逐步缩减表格，同时保留关键证据，因此将列选择和行筛选作为主要的缩减步骤。其余细粒度的格式处理和数值计算则统一交由代码生成，从而使工具集保持精简，同时仍支持复杂计算，并提供可验证、可追溯的输出。

3.2.1 确定性数据框操作符。 执行层通过具体的表格转换来落实推理，而非仅凭语言进行猜测。一个简洁的理由是，确定性转换不会增加关于答案的信息，

$$I(A; D' \mid Q) \le I(A; D \mid Q), \tag{17}$$

保留两个快照则能保留信息，

$$H(A \mid D, D', Q) = H(A \mid D, Q). \tag{18}$$

其中，A 表示答案，Q 表示问题，D 表示当前表格，D' 表示经过确定性转换后的表格。图2中的三个工具实现了这些映射：SelectColumns将模式投影到最小列集合，FilterRows评估受保护的谓词，而GenExeCode运行沙箱化 Python以计算定制统计量。每个工具都会输出上述元组，因此规划器可以看到更新后的快照和预览。

3.2.2 类型化接口与验证。 大语言模型提出的内容首先由工具调用规范化器进行规范化，该规范化器会强制转换 JSON 载荷并补齐缺失的标识符。随后，每个确定性工具都会在执行前应用模式感知防护——检查列是否存在、谓词有效性、类型/格式强制转换以及沙箱约束——并明确返回状态码和诊断信息。每次调用都会返回丰富的执行元数据（行列计数、哈希值、预览），以便推理层计算收缩、检测退化并编写反思提示；同一组数据也会传递给存储层进行版本管理。图2中的不良状态–动作对面板展示了验证失败时如何记录签名。为简洁起见，算法4端到端地概述了这些约定。SelectColumns 会执行不区分大小写的列匹配，并返回包含行列计数和预览的版本化快照。FilterRows 会规范化谓词并

算法4 类型化工具契约。每次调用都会消耗一个经过验证的快照，并输出一个带版本的后继。

SelectColumns$(table_uri, columns[\,])$
  $D \leftarrow$LoadDF$(table_uri)$
  $columns_{\text{actual}} \leftarrow$ GuardColumns$(D,\ columns)$
  $D' \leftarrow D[columns_{\text{actual}}]$
  返回 $(uri',\ hash',\ rows(D')\ cols(D')\ preview,\ status)$

FilterRows$(table_uri, condition)$
  $D \leftarrow$LoadDF$(table_uri)$
  $D_{\text{safe}} \leftarrow$类型规范化$(D)$
  $D' \leftarrow$ 评估谓词$(D_{\text{safe}},\ condition)$
  返回 $(uri',\ hash',\ rows(D')\ cols(D')\ preview,\ status)$

GenExeCode$(table_uri, instruction, context)$
  $D \leftarrow$LoadDF$(table_uri)$
  $code \leftarrow$ 大语言模型生成$(instruction, context, D)$
  $D' \leftarrow$ 沙箱执行$(code, D)$
  质量保护机制(D')
  返回 $(uri', hash', rows(D')\ cols(D')\ preview, code, status)$

类型，然后应用查询，接着输出筛选后的快照及元数据。当选择和筛选不足以完成任务时，GenExeCode 会生成并执行 Python 代码来进行聚合或多步算术运算，并返回带有执行元数据的结果快照。

3.2.3 执行与反思收敛。 由于每次模拟都恰好执行一次确定性工具调用，随后应用反思算子，因此每次迭代的成本主要由第 3.2 节所述的执行层操作决定。在标准 UCB 假设（奖励有界且持续探索）以及反思分数一致的条件下，TabTracer 的选择规则满足

$$\lim_{t\to\infty} P(\pi_t = \pi^*) = 1, \tag{19}$$

$$\mathbb{E}\left[V^{\pi^*}(s_0) - V^{\pi_t}(s_0)\right] = O\left(\sqrt{\frac{\log t}{t}}\right).$$

经典的 UCB1 置信半径可直接应用于此。即时反思能够在给定观测到的状态转移的条件下产生无偏的价值估计（由式 (3) 得出），而单步扩展可避免模拟偏差。将这些要素代入标准的 UCB1 遗憾分解，即可得到该界限。这一结果表明，TabTracer 的单步执行与反思循环能够逐步逼近高质量策略，同时保持运行时间可预测；这意味着执行层（确定性工具 + 反思）保留了式 (19) 所隐含的探索—利用权衡。

3.3 存储层

这一层对应图2中最右侧的模块：执行层返回的每个元组都会被物化为带版本的快照和哈希索引缓存条目，并为无效的状态–动作对记录惩罚。它有双重作用——通过缓存快照复用已有证据，并通过去重和黑名单抑制冗余或低质量的扩展。接下来，我们将介绍证据缓存与复用，然后讨论记忆与惩罚管理。

3.3.1 证据缓存与复用。 每个快照都被物化为元组 v_i =(统一资源标识符$_i$、哈希$_i$、 μ_i)，其中 μ_i 记录行列计数和执行元数据。TabTracer 不存储完整轨迹，而是维护一个哈希索引缓存：每个语义哈希都指向其 Parquet URI、相应的元数据以及两个基于签名的过滤器。首先，动作去重集合会抑制来自同一状态的语义等价工具调用（避免重复扩展相同动作）。其次，黑名单会存储此前导致反思分数较低或产生无效输出的状态-动作签名；这些状态–动作对会受到惩罚，并在后续模拟展开中跳过。对于屡次失败的 $(h,\ a)$ 对，其惩罚分数会在后续模拟展开中充当软约束；而缓存的 URI 则让回退例程能够直接引用 v_i，无需重新运行工具链。

令 H_t 表示截至时间 t 的哈希索引缓存，其中 h 表示语义表格状态哈希。每当 TabTracer 执行 $(h,\ a)$ 且反思分数仍低于容差时，便会以惩罚权重 $\pi(h,\ a)$ 记录该对：

$$\mathcal{H}_{t+1} = \mathcal{H}_t \cup \big\{(h, a, \pi(h,a)) : \pi(h,a) < 0\big\}. \tag{20}$$

再次遇到相同的 h 查询时，会先查询 H_t：跳过带有累积惩罚的签名，而未探索的签名则可以复用已存储的快照。因此，哈希感知剪枝可避免重复进行低质量扩展，并将预算集中用于新颖且效用更高的操作。

除了剪枝，缓存还会向回退例程提供此前已解决的统一资源标识符。图2 描绘了这一行为：红色叉号表示已累积惩罚的签名（因此后续模拟展开会跳过它们），绿色勾号则表示无需重新运行工具即可解引用的快照。由于每个条目都保留了 Parquet 指针和表元数据，重新读取哈希只需解引用 v_i，而无需重放整个工具链。因此，存储层仍然十分轻量：它存储快照、惩罚和可复用的统一资源标识符，但不存储完整的奖励历史或后验参数。

3.3.2 内存管理。 仅分发类型正确的动作（指示符$\mathbf{1}_{\text{typed}}(a,\ D)$ = 1），并将任何失败签名（$h_t,\ a$）追加至黑名单 K，以便后续尝试无需重新运行工具便会被丢弃（其中 h_t 是当前状态的语义哈希）。存储层结合了惰性加载和基于哈希的缓存——若两个节点具有相同的语义哈希，则仅保留指针。诸如式（14）中的均值等增量统计量，可避免存储完整的奖励历史。当低效用分支的 UCB1 分数低于

$$\xi_t = \max_{n \in \text{frontier}} \mathrm{UCB1}(n,t) - \kappa, \tag{21}$$

时进行剪枝，其中 κ 经过校准，以确保树保持在固定的内存预算范围内。

4 实验

本节介绍 TabTracer 的实验评估。我们首先介绍实验中使用的数据集、基线方法和评估指标。随后，我们报告 TabTracer 相较于强基线方法在整体有效性和成本效益方面的结果。此外，我们还分析了算术密集型工作负载下的数值鲁棒性，进行了工具级消融实验，并进一步考察了表格大小和搜索深度对可扩展性的影响。

表1：实验中使用的数据集。“数值型”表示标准答案包含数值的查询所占百分比。表格（S/M/L）对应表格大小 <500、500–1500，以及 >1500 词元。

数据集	查询数量	数值型	答案类型	表格（小/中/大）
TabFact	2,024	–	是 / 否	1188 / 813 / 23
WikiTQ	4,344	51.3%	字符串 / 数字 / 日期	2006 / 1867 / 471
CRT	728	28.0%	自由形式（文本 +数值型）	499 / 221 / 8

4.1 实验设置

4.1.1 数据集。我们在 TabFact、WikiTQ 和 CRT 的官方测试集上评估 TabTracer。表 1 展示了所用数据集的统计结果。TabFact 包含 2,024 个表格-声明对，每对都以二元标签（是/否）作答。WikiTQ 涵盖 4,344 个开放领域问题，答案可以是字符串、数字或日期。CRT 包含 728 个开放式任务，涉及事实型和分析型查询，其中 204 个（28.0%）需要进行显式数值计算。

4.1.2 系统配置。

实验环境。 实验在 Linux 6.6.80 主机上运行，该主机配备 16 核 Intel Xeon Gold 6133 CPU 和 31 GiB 内存。Qwen2.5-72B-Instruct [39] 通过 vLLM [24] 在 8 张 NVIDIA V100 GPU（CUDA 12.1）上提供服务。Qwen3-32B [40] 使用 8 张 NVIDIA L40 GPU（CUDA 12.1）。Qwen3-32B-Direct 禁用提供方侧的思考 模式，因此作为纯指令跟随模型运行，不使用辅助推理令牌。GPT-4.1-mini [28] 通过 OpenAI API 调用。该实现依赖 PyTorch 2.1 [32],、langgraph 0.5.0、langchain 0.3.26 和 pandas 2.3.0。

TabTracer 实现。 每个节点生成三个候选扩展，将搜索深度限制为 5，设置模拟预算为 15，并将 LangGraph 递归限制设为 50。默认启用基于哈希的状态缓存和回滚。候选生成和反思的基础温度设为 0.1。

评估指标。 我们报告精确匹配（EM）、输入/输出 token 数量、工具成功率（SR）、工具采用率（AR，即工具调用中其输出仍保留在保留分支上的比例）、状态复用率以及平均模拟次数。我们遵循 Table-Critic 协议，在比较前对标点符号、格式和时间戳进行归一化。对于 WikiTQ 和 CRT，我们在归一化后比较答案，而 TabFact 则直接使用精确匹配。

4.1.3 基线方法。 我们选择了九个全面的基线模型进行比较。所有基线模型均在相同的实验条件（模型版本、硬件和评估指标）下进行评估，以确保比较公平。我们通过系统的网格搜索，在验证集上调整超参数，以优化性能，同时保持计算预算具有可比性。

基于提示的推理方法：

- **端到端问答**：直接回答问题，不经过中间步骤；使用标准的少样本提示，每个数据集提供 0 个示例。
- **少样本问答**：通过 3–5 个精心挑选的示例增强少样本提示，涵盖不同的问题类型和表格结构。

• **思维链** [45]：针对表格推理任务，采用标准思维链提示进行逐步推理。
• **CoT-Consistent** [43]：通过 $k = 5$ 个样本进行自一致性思维链，并采用多数投票选择最终答案。
• **Lotus** [33]：语义算子框架，支持声明式 DataFrame 操作并提供统计准确性保证。我们实现了核心语义算子（sem_filter、sem_join、sem_topk），并整合了针对金融和科学模式的领域特定约束。

基于智能体的推理系统：

• **Chain-of-Table** [44]：确定性的转换程序，操作分类器已在我们的数据集上重新训练。程序长度最多限制为 12 个操作，以防止无限循环。

• **Reactable**[53]：采用 ReAct 风格智能体，通过迭代的推理—行动循环运行。配置包含完整的工具套件（筛选、排序、聚合、连接），并使用束搜索（束宽 = 4）选择操作。

• **MACT** [58]：采用规划器、执行器和验证智能体的多智能体协作框架。我们采用推荐的两轮审议协议，并在所有智能体间共享工具适配器，以确保一致性。

• **Table-Critic** [50]：采用三个批判-改进循环的迭代式自我批判系统。它使用与 Chain-of-Table 相同的算子集来执行工具。

4.2 与基线方法的比较

我们在三个表格推理数据集（TabFact、WikiTQ、CRT）上将 TabTracer 与多种基线方法进行比较，展示了其有效性和效率。

4.2.1 有效性评估。表 2 显示，TabTracer 的表现显著优于最先进的基线方法，在不同数据集和骨干模型上均取得持续且稳定的高精确匹配得分，并且在几乎所有配置中都名列前茅。具体而言，在复杂的 CRT 数据集上，TabTracer 取得了显著提升，相较于 Table-Critic（Qwen3-32B-Direct）最高提升 7.4%，且在其他大语言模型骨干模型上也持续取得提升。即使在较简单的事实验证数据集（TabFact）上，TabTracer 也提升了 0.2–1.7%，展现出稳定的性能。

基于提示的基线方法在某些设置下仍具竞争力，尤其是在事实型查询上。CoT-Consist 是表 2 中最强的基于提示的基线方法。它沿用清晰的思维链推理路径，并通过对多条推理链进行投票来提升鲁棒性。然而，它仍依赖纯语言检查，这会影响其在数值任务上的表现。基于智能体的方法会调用工具以减少仅由模型导致的幻觉，而 Table-Critic 是最强的基于智能体的基线方法。然而，大多数智能体仍采用线性轨迹或较弱的步骤验证，因此错误可能逐步传播，且结果在不同骨干模型间的差异更大。TabTracer 在各数据集和骨干模型上均排名第一或第二。它表现出色，是因为感知模式的子表切片减少了干扰项，步骤级验证能捕捉中间错误，而蒙特卡洛树搜索剪枝则会移除冗余分支，同时让搜索保持聚焦。

为探究推理大语言模型对表格推理的影响，我们在表 2 中比较了具有代表性的推理大语言模型。可以看出，现有的基于提示的方法和基于智能体的方法表现出很强的对大语言模型骨干模型的敏感性。例如，与 Qwen3-32B 相比，CoT-Consist 和 Chain-of-Table 在 Qwen3-32B-Direct 上的性能显著下降，凸显了它们对模型级推理能力的依赖。相比之下，TabTracer 在不同模型类型上均保持稳健，始终取得最佳或次佳表现。

4.2.2 效率评估。表 3 比较了各方法的令牌消耗。基于提示的系统效率最高，因为它们只需一次运行或少量采样，无须反复调用工具。工具密集型智能体的成本最高，因为迭代式工具执行会反复引入大型表格上下文。基于树的智能体则介于这两种极端之间。它们通过多轮批评提升准确性，但也增加了额外调用和重复上下文。MACT 是一个特殊情况：输入令牌较少，但输出令牌较多。即使输入较少，多智能体对话也会推高输出令牌数。TabTracer 是智能体系统中性价比最高的。子表切片可尽早缩小上下文，蒙特卡洛树搜索剪枝会移除冗余分支，而单步执行与反思循环则能在保持准确性的同时控制预算。

跨骨干模型稳定性分析表明，对于基于工具的智能体，不同大语言模型的 Token 用量可能存在很大差异。最明显的对比出现在 Qwen3-32B 与 Qwen3-32B-Direct 之间，思考模式会生成长得多的输出。每一步都生成长篇推理文本的方法会进一步放大这一差距，因为每次工具调用都会包含额外的模型输出。TabTracer 在不同骨干模型间的差异较小。它限制每个查询的工具调用次数，并使每步输出简短且结构化，因此骨干模型冗长程度的影响较小。这表明，我们观察到的稳定性源于方法设计，而非某个特定的骨干模型。

4.3 数值鲁棒性

数值鲁棒性对表格推理至关重要：算术幻觉很容易在链式运算中传播，并导致最终决策发生逆转。TabFact 的整体表现已体现出事实忠实度的提升。为单独考察数值幻觉，我们从 WikiTQ 和 CRT 中提取了算术密集型子集进行重点分析。表 4 评估了 TabTracer 在 WikiTQ 和 CRT 的算术密集型子集上抑制数值幻觉的能力。我们从三个维度分析 TabTracer 的表现：特定任务的准确性提升、跨模型稳定性，以及与基线方法的比较。

基线方法在数值推理方面存在明显局限。基于提示的方法在 CRT 上停滞于 45%，表明更长的文本推理链仍难以实现精确计算。不具备迭代精炼能力的智能体系统（如 Chain-of-Table、MACT）在 CRT 上仍低于 47%；而 Table-Critic 是表现最强的基线方法，在 CRT 上达到 41–47%，在 WikiTQ 算术子集上约为 76–84%。TabTracer 的表现大幅优于这些基线方法。在 CRT 算术查询上，它在不同骨干模型上的准确率达到 53–57%，比 Table-Critic 高出 8.9–12.4 个百分点。在 WikiTQ 算术子集上，提升幅度较小但稳定，为 0.3–2.5 个百分点。这些结果直接回应了数值幻觉问题。通过单独考察算术密集型子集，我们表明 TabTracer 减少了数值幻觉，而不仅仅是提升整体精确匹配率。性能提升源于基于执行的精炼。工具—反思

表 2：表格问答数据集上的实验结果，其中结果显示精确匹配准确率。粗体数字表示最佳结果，下划线数字表示次优结果。

方法	Qwen3-32B-Direct			Qwen2.5-72B			Qwen3-32B			GPT-4.1-Mini		
	TabFact	WikiTQ	CRT	TabFact	WikiTQ	CRT	TabFact	WikiTQ	CRT	TabFact	WikiTQ	CRT
基于提示的推理												
端到端问答	0.7915	0.5615	0.4904	0.8557	0.6271	0.5275	0.9407	0.7585	0.6093	0.8597	0.6634	0.5508
少样本问答	0.8187	0.5930	0.5137	0.8439	0.5907	0.4835	0.9303	0.7742	0.6058	0.8295	0.5654	0.4629
思维链	0.9185	0.5413	0.4698	0.9175	0.7671	0.6181	0.9298	0.7546	0.6202	0.9437	0.7581	0.5865
CoT-Consist	0.9338	0.5927	0.4959	0.9358	0.7828	0.6305	0.9447	0.7789	0.6168	**0.9491**	0.7726	0.6016
Lotus	0.8434	0.5750	0.4533	0.8869	0.6722	0.5659	0.9224	0.7516	0.6140	0.8908	0.7319	0.5659
基于智能体的系统												
Chain-of-Table	0.8864	0.4918	0.4317	0.9077	0.7231	0.5604	0.9030	0.7540	0.4684	0.8664	0.6237	0.5483
Reactable	0.8063	0.6001	0.5618	0.6764	0.5320	0.3571	0.8167	0.5992	0.5591	0.6809	0.4936	0.2816
MACT	0.8706	0.7120	0.5824	0.8923	0.7306	0.5934	0.9422	0.7957	0.6621	0.8923	0.6685	0.5797
Table-Critic	0.9430	0.8301	0.6414	0.9260	0.7770	0.6250	0.9417	0.8285	0.6428	0.9402	0.8128	0.6634
TabTracer（我们的方法）	**0.9447**	**0.8503**	**0.7157**	**0.9373**	**0.8126**	**0.7047**	**0.9486**	**0.8384**	**0.7074**	0.9452	**0.8357**	**0.7143**

表3：各数据集的 Token 用量（输入令牌 / 输出令牌）。

方法	Qwen3-32B-Direct						Qwen2.5-72B					
	TabFact		WikiTQ		CRT		TabFact		WikiTQ		CRT	
	输入	输出	输入	输出	输入	输出	输入	输出	输入	输出	输入	输出
CoT-Consist	3634.02	919.18	4966.36	832.27	3494.43	1410.83	3624.04	**839.23**	4928.08	876.33	3474.43	1535.82
Chain-of-Table	15212.50	1160.23	9010.26	550.63	9587.52	**386.70**	12656.28	907.94	8665.55	**447.27**	7701.82	**370.02**
Reactable	7250.02	680.63	19091.73	663.83	8633.00	1022.57	21530.03	864.07	21530.03	864.07	12832.69	1337.33
MACT	**1140.78**	2232.01	**2307.44**	1391.60	**2177.00**	2188.53	**1136.78**	1708.78	**2303.43**	1371.15	**2173.00**	2044.58
Table-Critic	9531.60	366.20	12032.00	**528.70**	13337.60	796.30	23559.30	1713.30	29660.70	1653.10	26571.20	1151.90
TabTracer（我们的方法）	1983.00	**251.40**	6650.40	685.80	4373.80	533.40	2913.50	911.70	7759.60	757.30	4926.10	940.10

方法	Qwen3-32B						GPT-4.1-Mini					
	TabFact		WikiTQ		CRT		TabFact		WikiTQ		CRT	
	输入	输出	输入	输出	输入	输出	输入	输出	输入	输出	输入	输出
CoT-Consist	3715.40	**954.12**	4918.51	2323.75	3464.43	3992.56	3157.56	710.49	4298.59	773.76	3072.58	1357.60
Chain-of-Table	11677.74	8976.12	5559.84	2338.02	7546.68	6655.20	12880.67	1169.83	12170.07	924.73	10806.01	**581.89**
Reactable	7537.09	3849.19	19142.75	3939.10	8761.14	4321.98	8696.00	2184.04	29323.53	2153.97	13997.54	2323.82
MACT	**1136.78**	4510.92	**2303.40**	3947.27	**2173.00**	6346.85	**1028.51**	3784.13	**1995.81**	1176.87	**1418.05**	1392.37
Table-Critic	10337.80	1255.80	13123.00	2609.30	15495.50	2702.60	18378.20	**609.00**	22407.10	819.80	23457.30	1055.10
TabTracer（我们的方法）	1984.40	1752.20	6581.00	**1390.40**	4635.70	**1365.50**	1959.30	911.20	6831.80	**720.00**	5274.50	1182.90

表4：WikiTQ 和 CRT 上的数值型推理准确性（精确匹配）。样本数量：WikiTQ = 2229，CRT = 204。

方法	Qwen3-32B-Direct		Qwen2.5-72B		Qwen3-32B		GPT-4.1-Mini	
	WikiTQ	CRT	WikiTQ	CRT	WikiTQ	CRT	WikiTQ	CRT
基于提示的推理								
端到端问答	0.5034	0.2108	0.5509	0.2500	0.7856	0.4069	0.5998	0.2921
少样本问答	0.5334	0.2206	0.5352	0.1961	0.7878	0.3824	0.4773	0.1881
思维链	0.6123	0.2549	0.7511	0.3873	0.8054	0.4510	0.7984	0.4406
CoT-Consist	0.6831	0.2843	0.7712	0.4020	0.8228	0.4412	0.8153	0.4505
Lotus	0.5379	0.2451	0.6326	0.3092	0.7672	0.3921	0.6376	0.3333
基于智能体的系统								
Chain-of-Table	0.5283	0.3465	0.6659	0.3529	0.7798	0.4118	0.5422	0.2637
Reactable	0.5362	0.3333	0.4613	0.1176	0.5344	0.3627	0.4694	0.1485
MACT	0.6754	0.3235	0.7001	0.3186	0.8032	0.4608	0.6404	0.3366
Table-Critic	0.8362	0.4604	0.7649	0.4149	**0.8385**	0.4406	0.8205	0.4656
TabTracer（我们的方法）	**0.8398**	**0.5743**	**0.7873**	**0.5347**	0.8277	**0.5644**	**0.8250**	**0.5545**

循环能够纠正中间的算术错误，并迫使计算与表结构保持一致；这很难通过纯语言推理或单次执行实现。TabTracer 还展现出更强的跨模型稳定性。其 CRT 准确率保持在相差 4 个百分点（53.5–57.4%），而 Table-Critic 的变化幅度更大且数值较低（41.5–46.6%）。这种稳定性表明，TabTracer 的改进源于方法设计，而非特定的骨干模型。

4.4 可扩展性实验

本节考察 TabTracer 如何随表格规模变化而扩展。表 5 按表格大小（小型/中型/大型）对查询进行分层，并使用三级绿色渐变来突出不同数据集上的扩展趋势。

对于小型表格，TabTracer 在 Tab-Fact 上的准确性达到 94%–95%，在 WikiTQ 上达到 86%（与领先的基线方法 Table-Critic 持平），并在 CRT 上保持 3–6% 的优势。这表明，当表格紧凑且证据易于提取时，TabTracer 仍能保持较高的准确性。

随着复杂度提升至中型表格，TabTracer 在 WikiTQ 上仍保持约 84% 的稳定表现，而竞争方法则大幅退化（CoT-Consist 下降 15 个百分点，MACT 从 82% 降至 67%）；在 CRT 上，TabTracer 的准确性维持在 67%–70%，而 Chain-of-Table 则下降 12%。这

表 5：不同数据集和模型配置下小型/中型/大型准确性对比（精确匹配）；T. 表示 TabFact，W. 表示 WikiTQ，C. 表示 CRT。

方法	Qwen3-32B-Direct									Qwen2.5-72B								
	小型			中型			大型			小型			中型			大型		
	T.	W.	C.	T.	W.	C.	T.	W.	C.	T.	W.	C.	T.	W.	C.	T.	W.	C.
端到端问答	0.81	0.62	0.51	0.76	0.52	0.45	0.83	0.45	0.25	0.86	0.68	0.57	0.84	0.60	0.45	0.83	0.49	0.25
思维链	0.93	0.60	0.50	0.88	0.51	0.42	0.91	0.37	**0.62**	0.93	0.81	0.66	0.91	0.76	0.55	0.96	0.61	0.50
CoT-Consist	0.94	0.65	0.53	0.92	0.57	0.43	0.96	0.42	**0.62**	**0.94**	0.82	0.66	0.93	0.78	0.58	**1.00**	0.65	0.50
Chain-of-Table	0.90	0.53	0.55	0.86	0.52	0.45	**1.00**	0.45	0.25	0.91	0.76	0.60	0.89	0.70	0.48	**1.00**	0.56	0.38
MACT	0.88	0.76	0.61	0.85	0.69	0.53	0.96	0.61	0.50	0.91	0.78	0.62	0.87	0.70	0.54	0.87	0.63	0.50
Table-Critic	0.94	0.86	0.68	**0.95**	0.84	0.56	0.96	0.68	0.50	0.93	0.82	0.66	0.93	0.76	0.54	0.96	0.64	0.50
TabTracer（我们的）	**0.94**	**0.87**	**0.72**	0.94	**0.84**	**0.70**	**1.00**	**0.79**	**0.62**	0.94	**0.84**	**0.72**	**0.93**	**0.81**	**0.67**	**1.00**	**0.73**	**0.62**

方法	Qwen3-32B									GPT-4.1-Mini								
	小型			中型			大型			小型			中型			大型		
	T.	W.	C.	T.	W.	C.	T.	W.	C.	T.	W.	C.	T.	W.	C.	T.	W.	C.
端到端问答	**0.95**	0.81	0.62	0.93	0.75	0.57	0.96	0.61	0.62	0.87	0.72	0.57	0.84	0.64	0.52	0.87	0.51	0.38
思维链	0.93	0.78	0.64	0.93	0.76	0.57	**1.00**	0.61	**0.75**	0.94	0.79	0.63	0.94	0.75	0.53	**1.00**	0.62	**0.62**
CoT-Consist	0.94	0.80	0.63	0.95	0.79	0.58	**1.00**	0.64	**0.75**	**0.95**	0.80	0.63	**0.95**	0.78	0.54	**1.00**	0.63	**0.62**
Chain-of-Table	0.92	0.77	0.64	0.89	0.76	0.57	0.91	0.64	0.50	0.87	0.65	0.59	0.86	0.61	0.45	0.95	0.56	**0.62**
MACT	0.93	0.82	0.68	**0.96**	0.79	0.61	0.96	0.67	0.62	0.90	0.72	0.61	0.87	0.64	0.51	**1.00**	0.57	0.50
Table-Critic	0.94	0.86	0.67	0.94	**0.84**	0.60	0.96	0.69	0.50	0.94	0.84	0.70	0.94	0.82	0.57	**1.00**	0.68	**0.62**
TabTracer（我们的方法）	0.95	**0.86**	**0.72**	0.94	0.83	**0.67**	**1.00**	**0.77**	0.50	0.94	**0.86**	**0.73**	0.95	**0.83**	**0.68**	**1.00**	**0.75**	0.50

表明，与单轨迹方法相比，TabTracer 具有更强的韧性，因为树搜索机制能够实现……
探索多条路径，以应对复杂度的提升。

在大型表格上，TabTracer 在 WikiTQ 上保持 73–79% 的准确性，比 Table-Critic 的 64–68% 高出 7–12 个百分点，并且在 CRT 上对各种开放权重模型均保持 12% 的优势。尽管性能自然有所下降，TabTracer 的树搜索和反思机制在高复杂度场景中仍展现出显著优势。

表 5 使用三级绿色渐变，较深的单元格表示在较小表格上的准确性较高，较浅的单元格则表示在较大表格上的准确性较低。颜色趋势表明，在不同方法中，随着表格从小型变为大型，准确性始终呈下降趋势，这证实了大型表格更难处理。在 WikiTQ 和 CRT 上，这种下降最为明显；而 TabFact 的变化较为缓和，因为它是一项二元验证任务。我们还注意到，TabFact 和 CRT 的大型表格子集相对较小，因此它们在大型表格上的结果可能波动更大。与基线方法相比，TabTracer 在 WikiTQ 和 CRT 上从小型表格到大型表格的准确性降幅更小，而基于提示的方法和单轨迹智能体的性能下降更为显著。这表明，随着表格大小增加，TabTracer 具有更强的可扩展性。

4.5 消融研究

我们通过在 CRT 上进行单组件消融实验，验证了每种工具的必要性，并衡量其对准确性和令牌效率的影响，如图 3 所示。移除任何工具都会降低精确匹配准确率。当规划器无法委派任务时，词元用量也会增加。移除 SelectColumns 时，增幅最大；而移除 GenExecCode 后，尽管准确性有所下降，词元用量仍增加约 1k。

移除 SelectColumns 会导致最严重的性能下降：精确匹配准确率从 70.5% 骤降至 52–53%（下降 17–19

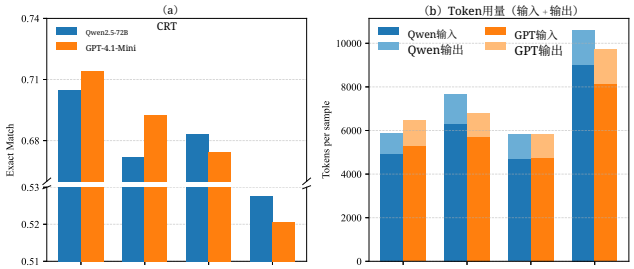


图 3：CRT 上的工具消融实验。我们报告移除各个操作符后的精确匹配准确率和 Token用量。

个百分点），其准确性影响在全部三个工具中最大。同时，对于 Qwen2.5-72B，输入令牌消耗从 4.9k 增至 9.0k 个词元，计算预算几乎翻倍。这种双重影响源于 SelectColumns 在缩减表格宽度方面的作用——如果不进行列剪枝，后续每次工具调用都会携带完整的表格宽度，从而增加上下文长度，并迫使模型针对无关列进行推理。因此，SelectColumns 成为兼顾准确性与效率的关键组件。

移除 FilterRows 会使准确性降低 2–4 个百分点，同时令词元消耗增加约 600 个词元。FilterRows 可通过行级过滤消除干扰信息；缺少这一功能时，系统就不得不在更大的行空间中搜索，从而增加推理难度。虽然其单独影响小于 SelectColumns，但在多步推理链中累积起来仍然影响显著。

移除 GenExecCode 会使准确性降低 2–4 个百分点，但反常的是，词元消耗会增加约 $1k$ 个词元。出现这种违背直觉的情况，是因为缺乏代码生成能力时，系统会转而采用更冗长的自然语言描述并进行多次工具调用，以完成复杂计算；因此，尽管准确性降低，词元消耗反而膨胀。由此可见，GenExecCode 提供了简洁高效的计算路径，优于自然语言替代方案。

完整工具集在准确性和效率两个方面均实现了最佳性能。移除任何单个工具都会降低准确性或增加令牌消耗，这证明了架构中每个工具的必要性。

4.6 组件分析

本分析探究了步骤级可靠性和搜索行为；若缺少中间验证，可能会发生错误传播。因此，我们报告工具成功率/采用情况以及搜索诊断信息。

图4-5-6为本组件分析提供了依据：我们先考察运行稳定性（工具成功率和采用情况），再检查搜索行为（状态复用和模拟次数）。

图4显示，在各个模型中，每种工具的首次尝试成功率均超过99。7%。FilterRows 和 SelectColumns 的成功率达到99。9–100%，这表明即使是复杂筛选器也能确定性地执行；而 GenExecCode 即使需要合成定制的 Python 代码，成功率仍保持在99。7–100%。

首次尝试的采用率反映了任务难度。在 WikiTQ 上使用 Qwen3-32B-Direct 时，FilterRows 保留了 77.5% 的输出；在 CRT 上则为 70.2%。但在 WikiTQ 上使用 Qwen2.5-72B 时，采用率降至 27.6%，因为模型会探索更广泛的分支。后续尝试至关重要：第二次调用额外找回了 8.8% 的答案，第三次又进一步找回了 14.7%。相比之下，GPT-4.1-mini 在 CRT 上的起始采用率较高（65.2%），并借助效果显著的第二次尝试（53.7%）弥补了剩余差距。SelectColumns 几乎始终表现完美（首次尝试采用率为 ≥98.9%），证实列选择已成为一项已解决的子任务。

我们提出的基于蒙特卡洛树搜索的架构展现出较高的运行效率，验证了采用树引导探索而非穷举推理的选择。图5显示，TabTracer 的模拟次数适中（每个查询 2.83–3.86 次），同时最多可复用 47.7% 的缓存状态，表明缓存策略能够有效分摊搜索成本。如图6所示，64–88% 的查询在三次模拟内收敛，而大规模搜索（>6 次扩展）占比低于 11%，体现了对探索过程的良好控制。这些结果验证了该搜索架构能够在探索深度与计算效率之间实现平衡。这是因为基于反思的评分引导选择，而基于哈希的过滤则会移除重复的状态-动作对，从而及早剪除冗余分支。

5 结论

本文介绍 TabTracer，这是一种蒙特卡洛树搜索智能体，旨在为复杂表格推理恢复严谨性。我们首先将规划与执行解耦：信息增强型 UCB1 策略仅在中间问题有望带来新证据时才扩展工具调用，而即时反思则无需进行冗长的模拟展开即可为每个分支评分。随后，我们以确定性数据框工具为推理提供依据，从而产生可审计的操作并抑制幻觉。最后，TabTracer 会持久化带版本的表格状态，以便以较低成本进行回溯和状态复用，从而控制探索预算。

参考文献

[1] Nikhil Abhyankar、Vivek Gupta、Dan Roth 和 Chandan K Reddy。2025年。H-star：大语言模型驱动的表格混合 SQL-文本自适应推理。载于《2025年美洲计算语言学协会分会会议论文集：人类语言技术》（第1卷：长篇论文）。8841–8863。[2] Erik Altman、Jovan Blanuša、Luc Von Niederhäusern、Béni Egressy、Andreea Anghel 和 Kubilay Atasu。2023年。用于反洗钱模型的逼真合成金融交易数据。《神经信息处理系统进展》36（2023年），29851–29874。[3] Vadim Borisov、Kathrin Sessler、Tobias Leemann、Martin Pawelczyk 和 Gjergji Kasneci。 [无日期]。《语言模型是逼真的表格数据生成器》。载于第十一届国际学习表征会议。[4] Cameron B Browne、Edward Powley、Daniel Whitehouse、Simon M Lucas、Peter I Cowling、Philipp Rohlfshagen、Stephen Tavener、Diego Perez、Spyridon Samoth-rakis 和 Simon Colton。2012年。蒙特卡洛树搜索方法综述。《IEEE计算智能与人工智能游戏汇刊》4卷1期（2012年），1–43。[5] Douglas Burdick、Marina Danilevsky、Alexandre V Evfimievski、Yannis Katsis 和 Nancy Wang。2020年。面向科学和企业应用的表格提取与理解。《VLDB基金会论文集》13卷12期（2020年），3433–3436。[6] Avi Caciularu、Alon Jacovi、Eyal Ben-David、Sasha Goldshtein、Tal Schuster、Jonathan Herzig、Gal Elidan 和 Amir Globerson。2024年。Tact：借助信息抽取工具推进复杂聚合推理。《神经信息处理系统进展》37（2024年），34775–34799。[7] Qingpeng Cai、Kaiping Zheng、HV Jagadish、Beng Chin Ooi 和 James Yip。2024年。CohortNet：赋能可解释医疗保健分析中的队列发现。《VLDB基金会论文集》17卷10期（2024年），2487–2500。[8] Dingwei Chen、Ziqiang Liu、Feiteng Fang、Chak Tou Leong、Shiwen Ni、Ahmadreza Argha、Hamid Alinejad-Rokny、Min Yang 和 Chengming Li。2025年。先扩展，再推断：通过前置层插值提升大型语言模型的事实性。arXiv预印本 arXiv:2506.02973（2025年）。[9] Tianyi Chen、Jun Gao、Hedui Chen 和 Yaofeng Tu。2023年。Loger：一种旨在生成高效稳健查询执行计划的学习型优化器。《VLDB基金会论文集》16卷7期（2023年），1777–1789。[10] Wenhu Chen。2023年。大型语言模型是少样本（1-shot）表格推理器。载于《计算语言学协会发现：EACL 2023》。1120–1130。[11] Ziru Chen、Michael White、Ray Mooney、Ali Payani、Yu Su 和 Huan Sun。2024年。何时适合用树搜索进行大语言模型规划？这取决于判别器。载于《计算语言学协会第62届年会论文集》（第1卷：长篇论文）。13659–13678。[12] Zhoujun Cheng、Tianbao Xie、Peng Shi、Chengzu Li、Rahul Nadkarni、Yushi Hu、Caiming Xiong、Dragomir Radev、Mari Ostendorf、Luke Zettlemoyer 等。 [无日期]。《以符号语言约束语言模型》。载于第十一届国际学习表征会议。[13] Xi Fang、Weijie Xu、Fiona Anting Tan、Ziqing Hu、Jiani Zhang、Yanjun Qi、Srinivasan H Sengamedu 和 Christos Faloutsos。 [无日期]。《表格数据中的大型语言模型（LLM）：预测、生成与理解——综述》。《机器学习研究汇刊》（[无日期]）。[14] Yury Gorishniy、Ivan Rubachev、Valentin Khrulkov 和 Artem Babenko。2021年。重新审视面向表格数据的深度学习模型。《神经信息处理系统进展》34（2021年），18932–18943。[15] Taicheng Guo、Xiuying Chen、Yaqi Wang、Ruidi Chang、Shichao Pei、Nitesh V Chawla、Olaf Wiest 和 Xiangliang Zhang。2024年。基于大型语言模型的多智能体：进展与挑战综述。载于第三十三届国际人工智能联合会议论文集。8048–8057。[16] Yu Guo、Shenghao Ye、Shuangwu Chen、Zijian Wen、Tao Zhang、Qirui Bai、Dong Jin、Yunpeng Hou、Huasen He、Jian Yang 等。2026年。重新思考表格问答中的表格剪枝：从顺序修订到黄金轨迹监督的并行搜索。arXiv预印本 arXiv:2601.03851（2026年）。[17] Immanuel Haffner 和 Jens Dittrich。2023年。利用启发式搜索高效计算连接顺序。《ACM数据管理论文集》1卷1期（2023年），1–26。[18] Jonathan Herzig、Pawel Krzysztof Nowak、Thomas Müller、Francesco Piccinno 和 Julian Eisenschlos。2020年。TaPas：通过预训练进行弱监督表格解析。载于计算语言学协会第58届年会论文集。4320–4333。[19] Aaron Hurst、Adam Lerer、Adam P Goucher、Adam Perelman、Aditya Ramesh、Aidan Clark、AJ Ostrow、Akila Welihinda、Alan Hayes、Alec Radford 等。2024年。GPT-4o系统说明。arXiv预印本 arXiv:2410.21276（2024年）。

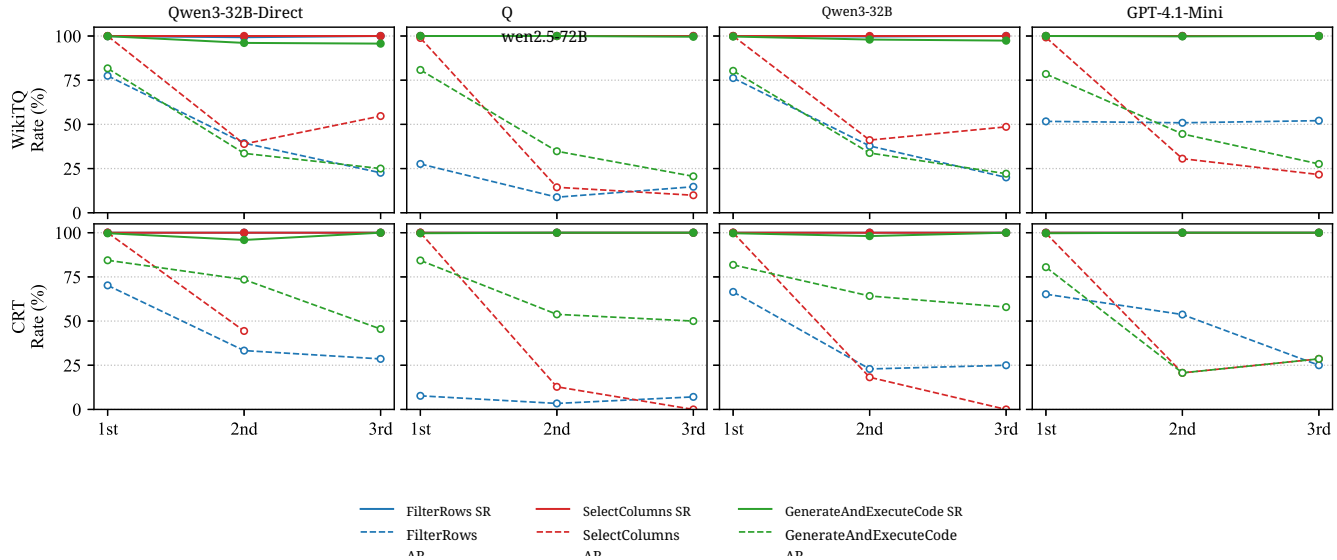


图4：各阶段的工具成功率（SR）和采用率（AR）。

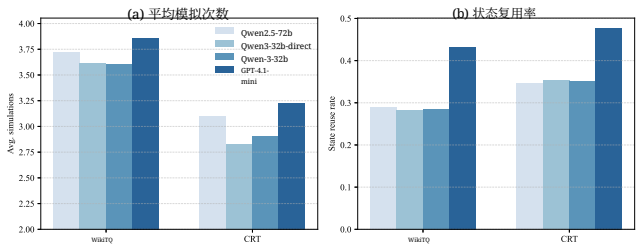
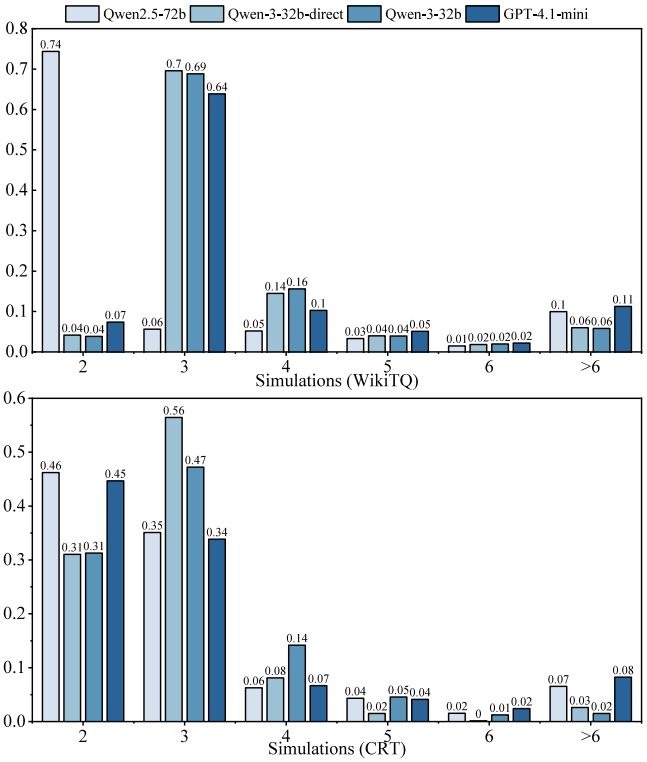


图5：WikiTQ 和 CRT 上的平均模拟次数与状态复用率。



Figu图6：WikiTQ 上模拟次数的分布 和 CRT。

[20] Sujay Kumar Jauhar、Peter Turney 和 Eduard Hovy。2016年。作为问答半结构化知识的表格。载于《计算语言学协会第*54*届年会论文集》（第*1*卷：长篇论文）。474–483。[21] Chuang Jiang、Mingyue Cheng、Xiaoyu Tao、Qingyang Mao、Jie Ouyang 和 Qi Liu。2025年。TableMind：用于工具增强型表格推理的自主程序化智能体。*arXiv*预印本 *arXiv:2509.06278*（2025年）。[22] Ruochun Jin、Xiyue Wang、Dong Wang、Haoqi Zheng、Yunpeng Qi、Silin Yang 和 Meng Zhang。2025年。TALON：用于长表格探索与问答的多智能体框架。载于《*2025*年自然语言处理实证方法会议论文集》。27385–27401。[23] Wenjun Ke、Ziyu Shang、Zhizhao Luo、Peng Wang、Yikai Guo、Qi Liu 和 Yuxuan Chen。2024年。通过强化学习和数据比例揭示大语言模型的事实性并注入知识。《中国科学：信息科学》67卷10期（2024年），209101。[24] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph E. Gonzalez、Hao Zhang 和 Ion Stoica。2023年。借助PagedAttention高效管理大语言模型服务的内存。载于《第*29*届*ACM SIGOPS*操作系统原理研讨会论文集》。[25] Peng Li、Yeye He、Dror Yashar、Weiwei Cui、Song Ge、Haidong Zhang、Danielle Rifinski Fainman、Dongmei Zhang 和 Surajit Chaudhuri。2024年。Table-GPT：面向多种表格任务的表格微调GPT。《*ACM*数据管理论文集》2卷3期（2024年），1–28。[26] Xiao Liu、Hao Yu、Hanchen Zhang、Yifan Xu、Xuanyu Lei、Hanyu Lai、Yu Gu、Hangliang Ding、Kaiwen Men、Kejuan Yang 等。[无日期]。AgentBench：将大语言模型作为智能体进行评估。载于第十二届国际学习表征会议。

[27] Md Nahid 和 Davood Rafiei。2024年。TabSQLify：通过表格分解增强大语言模型的推理能力。载于*2024*年计算语言学协会北美分会年会：人类语言技术（第*1*卷：长篇论文）论文集。5725–5737。[28] OpenAI。2025年。推出 API 中的 GPT-4.1。https://openai.com/index/gpt-4-1/。访问日期：YYYY-MM-DD。

[29] Paolo Papotti 和 Carsten Binnig。2025年。神经关系数据专题讨论：表格基础模型、大语言模型……还是两者兼有？VLDB基金会论文集 18，12（2025年），5513–5515。[30] Panupong Pasupat 和 Percy Liang。2015年。半结构化表格上的组合语义解析。载于计算语言学协会第53届年会和自然语言处理国际联合会议第7届会议论文集（第1卷：长篇论文）。1470–1480。[31] Panupong Pasupat 和 Percy Liang。2016年。从指称结果推断逻辑形式。载于计算语言学协会第54届年会论文集（第1卷：长篇论文）。23–32。[32] Adam Paszke、Sam Gross、Soumith Chintala、Gregory Chanan、Edward Yang、Zachary DeVito、Zeming Lin、Alban Desmaison、Luca Antiga 和 Adam Lerer。2017年。PyTorch 中的自动微分。（2017年）。[33] Liana Patel、Siddharth Jha、Melissa Pan、Harshit Gupta、Parth Asawa、Carlos Guestrin 和 Matei Zaharia。2025年。语义算子及其优化：在 LOTUS 中实现具有准确性保证的基于大语言模型的数据处理。VLDB基金会论文集 18，11（2025年），4171–4184。[34] Fotis Psallidas 和 Eugene Wu。2018年。Smoke：交互式速度下的细粒度数据沿袭。VLDB基金会论文集 11，6（2018年），719–732。[35] Krishna Puttaswamy、Abhijit Chakankar、Tao Tao、Zaheera Valani、Ramesh Chandra、William Chau、Mengxi Chen、Akram Chetibi、Tianyi Huang、Jonathan Keller 等。2025年。Delta Sharing：一种用于跨平台数据共享的开放协议。VLDB基金会论文集 18，12（2025年），5197–5209。[36] Shreya Shankar 和 Aditya G Parameswaran。2024年。使用 motion 构建响应式大语言模型管道。载于2024年国际数据管理会议伴随论文集。520–523。[37] Noah Shinn、Federico Cassano、Ashwin Gopinath、Karthik Narasimhan 和 Shunyu Yao。2023年。Reflexion：通过语言反馈强化学习的语言智能体。神经信息处理系统进展 36（2023年），8634–8652。[38] Yuan Sui、Mengyu Zhou、Mingjie Zhou、Shi Han 和 Dongmei Zhang。2024年。表格遇上大语言模型：大语言模型能理解结构化表格数据吗？一项基准测试与实证研究。载于第17届 ACM 国际网络搜索与数据挖掘会议论文集。645–654。[39] Qwen团队。2024年。Qwen2.5：一系列基础模型。https://qwenlm.github.io/blog/qwen2.5/ [40] Qwen团队。2025年。Qwen3技术报告。arXiv:2505.09388 [cs.CL] https://arxiv.org/abs/2505.09388[41] Ziyu Wan、Xidong Feng、Muning Wen、Stephen Marcus McAleer、Ying Wen、Weinan Zhang 和 Jun Wang。2024年。类AlphaZero树搜索可指导大语言模型的解码与训练。载于第41届国际机器学习会议论文集。49890–49920。[42] Lei Wang、Wanyu Xu、Yihuai Lan、Zhiqiang Hu、Yunshi Lan、Roy Ka-Wei Lee 和 Ee-Peng Lim。2023年。计划与求解提示：通过大语言模型提升零样本思维链推理。载于计算语言学协会第61届年会论文集（第1卷：长篇论文）。2609–2634。[43] Xuezhi Wang、Jason Wei、Dale Schuurmans、Quoc V Le、Ed H Chi、Sharan Narang、Aakanksha Chowdhery 和 Denny Zhou。2024年。自洽性提升语言模型中的思维链推理。载于第十一届国际学习表征会议。[44] Zilong Wang、Hao Zhang、Chun-Liang Li、Julian Martin Eisenschlos、Vincent Perot、Zifeng Wang、Lesly Miculicich、Yasuhisa Fujii、Jingbo Shang、Chen-Yu Lee 等。2024年。Chain-of-Table：在推理链中演进表格以实现表格理解。载于国际学习表征会议。[45] Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Fei Xia、Ed Chi、Quoc V Le、Denny Zhou 等。2022年。思维链提示促使大语言模型进行推理。神经信息处理系统进展 35（2022年），24824–24837。[46] An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv 等。2025年。Qwen3技术报告。arXiv预印本 arXiv:2505.09388（2025年）。[47] Shunyu Yao、Dian Yu、Jeffrey Zhao、Izhak Shafran、Tom Griffiths、Yuan Cao 和 Karthik Narasimhan。2023年。思维树：使用大语言模型审慎求解问题。神经信息处理系统进展 36（2023年），11809–11822。[48] Shunyu Yao、Jeffrey Zhao、Dian Yu、Nan Du、Izhak Shafran、Karthik R Narasimhan 和 Yuan Cao。 [n.d.]。ReAct：实现语言模型中推理与行动的协同。载于第十一届国际学习表征会议。[49] Pengcheng Yin、Graham Neubig、Wen-tau Yih 和 Sebastian Riedel。2020年。TaBERT：文本与表格数据联合理解的预训练。载于计算语言学协会第58届年会论文集。8413–8426。[50] Peiying Yu、Guoxin Chen 和 Jingjing Wang。2025年。Table-Critic：用于表格推理中协同批评与改进的多智能体框架。载于计算语言学协会第63届年会论文集（第1卷：长篇论文）。计算语言学协会，维也纳

，奥地利，17432–17451。

[51] Chi Zhang、Meihui Zhang、Yuxin Yang、Tao Chen 和 Zhaojing Luo。2025年。AixelAsk：大型表格问答的分步引导式检索与推理框架。*ACM*数据管理论文集 3，6（2025年），1–25。[52] Xuanliang Zhang、Dingzirui Wang、Longxu Dou、Baoxin Wang、Dayong Wu、Qingfu Zhu 和 Wanxiang Che。2024。FLEXTAF：通过灵活的表格格式增强表格推理。*arXiv*预印本 *arXiv:2408.08841*（2024）。[53] Yunjia Zhang、Jordan Henkel、Avrilia Floratou、Joyce Cahoon、Shaleen Deep 和 Jignesh M Patel。2024。ReAcTable：增强用于表格问答的 ReAct。*VLDB*基金会论文集 17，8（2024），1981–1994。[54] Yuchen Zhang、Panupong Pasupat 和 Percy Liang。2017。用于高效语义解析的宏语法与整体触发。载于*2017*年自然语言处理经验方法会议论文集。1214–1223。[55] Yusen Zhang、Ruoxi Sun、Yanfei Chen、Tomas Pfister、Rui Zhang 和 Sercan Arik。2024。智能体链：大语言模型在长上下文任务中的协作。神经信息处理系统进展 37（2024），132208–132237。[56] Andy Zhou、Kai Yan、Michal Shlapentokh-Rothman、Haohan Wang 和 Yu-Xiong Wang。2024。语言智能体树搜索统一了语言模型中的推理、行动与规划。载于第*41*届国际机器学习会议论文集。62138–62160。[57] Denny Zhou、Nathanael Schärli、Le Hou、Jason Wei、Nathan Scales、Xuezhi Wang、Dale Schuurmans、Claire Cui、Olivier Bousquet、Quoc V Le 等。 [未注明年份]。由简至繁的提示使大语言模型能够进行复杂推理。载于第十一届国际学习表征会议。[58] Wei Zhou、Mohsen Mesgar、Annemarie Friedrich 和 Heike Adel。2025年。借助工具使用进行高效多智能体协作，以实现复杂表格问答中的在线规划。载于计算语言学协会：*NAACL 2025*会议论文集。945–968。[59] Jun-Peng Zhu、Peng Cai、Kai Xu、Li Li、Yishen Sun、Shuai Zhou、Haihuang Su、Liu Tang 和 Qi Liu。2024。AutoTQA：借助多智能体大语言模型实现自主表格问答。*VLDB*基金会论文集 17，12（2024），3920–3933。[60] Yuchen Zhuang、Xiang Chen、Tong Yu、Saayan Mitra、Victor Bursztyn、Ryan A Rossi、Somdeb Sarkhel 和 Chao Zhang。2024。ToolChain*：利用 A*搜索在大语言模型中实现高效动作空间导航。载于国际学习表征会议。[61] Jin Ziqi 和 Wei Lu。2023。Tab-CoT：零样本表格思维链。载于计算语言学协会：*ACL 2023*会议论文集。10259–10277。