TableRAG：一种用于异构文档推理的检索增强生成框架

Xiaohan Yu[*]
Huawei Cloud BU, Beijing
yuxiaohan5@huawei.com

Pu Jian[*]
Huawei Cloud BU, Beijing
jianpu2@huawei.com

Chong Chen[†]
Huawei Cloud BU, Beijing
chenchong55@huawei.com

Abstract

增强检索生成（RAG）在开放域问答中表现出了相当的有效性。然而，当应用于包含文本和表格组件的异构文档时，现有的 RAG 方法显示出明显的局限性。将表格扁平化和块化策略打断了固有的表格式结构，导致信息丢失，并削弱了大模型在多跳、全局查询中的推理能力。为了解决这些问题，我们提出了 TableRAG，这是一种结合文本理解和对表格数据进行复杂处理的混合框架。TableRAG 以四个步骤反复操作：上下文敏感的查询分解、文本检索、SQL 编程和执行，以及组成中间答案生成。我们还开发了 HeteQA，这是一种用于评估多跳异构推理能力的新基准。实验结果表明，TableRAG 在公共数据集和我们的 HeteQA 上始终优于现有的基准，建立了异构文档问答的新技术标准。我们在 https://github.com/yxh-y/TableRAG/tree/main 发布了 TableRAG。

1 介绍

异构文档为基础的问题回答 (Chen et al., 2020)，需要对非结构化文本和结构化表格数据进行推理，具有相当大的挑战性。表格的行和列是相互依赖的，而自然语言文本是顺序的。在统一的 QA 系统中弥合这种差异仍然是一项艰巨的任务。

主要的方法扩展了检索增强生成（RAG）范式，其中表格被线性化为文本表示（例如，Markdown）(Gao et al., 2023; Jin and Lu, 2023; Ye et al., 2023b)。通常，使用区块策略 (Finardi et al., 2024)，其中被压平的表格被分割并与相邻的文本片段合并。在推理阶段，LLMs 根据检索到的前 N 个区块生成答案。然而，这些方法主要是为只需要对表格进行浅层理解的场景量身定制，例如直接答案提取 (Pasupat and Liang, 2015a; Zhu et al., 2021)。当应用于混合了文本和表格元素的庞大文档时，现有的 RAG 方法表现出关键的限制：

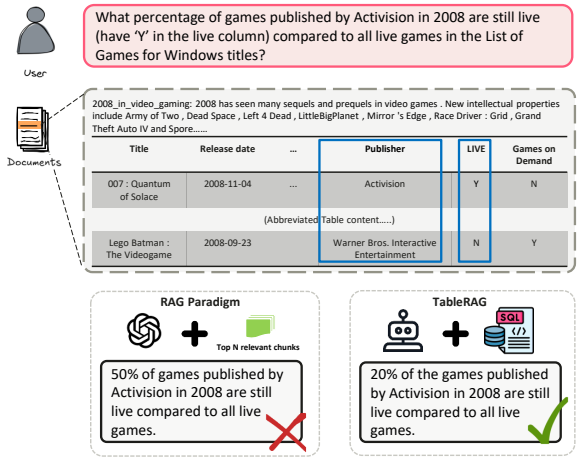

Figure 1: 一个基于异构文档的问答任务示例。

- 结构信息损失：表格结构完整性被破坏，导致信息丢失或不相关的上下文，阻碍下游大型语言模型的性能。
- 缺乏全局视角：由于文档的碎片化，RAG 系统在处理多跳全局查询 (Edge et al., 2024) 时存在困难，例如聚合、数学计算和其他需要对整个表格进行全面理解的推理任务。

如图 1 所示，RAG 方法计算的是前 N 个最相关区块的百分比，而不是整个表格，从而导致错误答案。

为了应对现有 RAG 系统的这些局限性，我们提出了 TableRAG，这是一种融合（SQL 执行和文本检索）的框架，能够在文本理解和复杂的表格数据处理之间动态转换。TableRAG 通过利用 SQL 作为接口与表格交互。具体而言，该框架通过两阶段过程运行：一个离线数据库构建阶段和一个在线推理阶段的迭代推理过程。迭代推理过程包括四个核心操作：（i）上下文敏感的查询分解，（ii）文本检索，（iii）SQL 编程和执行，以及（iv）中间答案生成。使用 SQL 可将与表格相关的查询视为不可分割的推理单元，从而实现精确的符号执行，增强计算效率和推理真实性。为了便于对异类文档上的多跳推理进行严格评估，我们引入了

[1]* These authors contributed equally to this work.
[2]† Corresponding author.

HeteQA，一个新颖的基准数据集，包含九个不同领域的 304 个示例。每个示例都包含五种不同的表格操作的组合。我们在现有的公共基准和我们的 HeteQA 数据集上评估 TableRAG，并与强大的基线进行比较，包括通用 RAG 和程序辅助的方法。实验结果表明，TableRAG 始终达到最先进的性能。总体而言，我们的贡献总结如下：在异类文档问答任务的背景下，我们将任务输入定义为广泛的文档，记为 $(\mathcal{T}, \mathcal{D})$，其中 $\mathcal{T}$ 表示文本内容，$\mathcal{D}$ 指代表格组件。给定一个用户问题 q，任务的目标是优化一个函数 $\mathcal{F}$，该函数在给定文本和表格上下文的情况下，可以生成正确的答案 $\mathcal{A}$。

2 TableRAG 框架

2.1 概述和设计原则

我们提出了 TableRAG，这是一个混合（SQL 执行和文本检索）框架，旨在维护表格的结构完整性并促进异质推理。如图 2 所示，TableRAG 包括离线和在线工作流程。离线阶段负责数据库构建，而在线阶段促进迭代推理。推理过程展开为四个阶段：(i) 上下文敏感查询分解，识别查询中文本和表格形式的各自角色。(ii) 文本检索。(iii) SQL 编程和执行，被选择性地调用以处理需要表格数据推理的子查询。(iv) 组成中间答案生成。优先使用 SQL 是因为它能够在结构化数据上利用符号执行的表达力，从而使用户查询中的表格组件被视为整体推理单元。相比之下，其他语言如 Python 在处理大规模数据或复杂工作负载时会产生大量计算开销 (Shahrokhi et al., 2024)。

在离线阶段，我们首先从异构文档中提取结构化组件，产生一组表格 $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_M\}$。为了实现信息检索，我们构建了两个平行语料库：一个文本知识库和一个表格模式数据库。文本知识库包括原始文本 $\mathcal{T}$ 和每个表格的 Markdown 渲染形式，表示为 $\hat{\mathcal{D}}$。$\hat{\mathcal{D}}$ 和 $\mathcal{T}$ 都被分割成块，然后使用预训练的语言模型 (Chen et al., 2024) 嵌入到密集的向量表示中。对于表格模式数据库的构建，我们通过以下模板定义每个表格 $\mathcal{D}_i$ 的标准化模式描述 $S(\mathcal{D}_i)$：然后，我们定义了从每个展平的表格块到其来源表格模式的映射：其中 $\hat{\mathcal{D}}_{i,j}$ 表示从表格 $\hat{\mathcal{D}}_i$ 派生的第 j 块。此映射确保局部片段保持与其派生的表结构的上下文关联。

这些表格也被输入到关系型数据库中（例如，MySQL [1]），支持后续在线推理中的符号查询执行。

为了处理需要对文本和表格进行组合推理的多跳、全局查询，我们引入了一个与方程 **??** 中的 $\mathcal{F}$ 一致的迭代推理过程。该过程包括四个核心操作：(i) 上下文敏感的查询分解，(ii) 文本检索，(iii) 程序生成并执行 SQL，以及 (iv) 组合中间答案生成。通过重复的分解和解析循环，逐步构建查询的解决方案。详细的提示模板在附录 E 中提供。

上下文敏感的查询分解 我们明确界定了文本和表格模式在推理过程中的各自角色。尽管一个与表格相关的查询可能涉及多个语义推理步骤，其表格解析可以简化为一个可执行操作。因此，有效的全局查询分解需要的不仅仅是简单的句法划分，还需要对底层数据源的结构意识。为此，我们首先从文本数据库中检索最相关的表内容，并通过映射函数 f 将其链接到对应的表模式描述 $S(\mathcal{D}^t)$。基于此，我们在 t 次迭代中形成一个子查询 q_t。

我们部署了一个检索模块，该模块在两个连续阶段内运行：向量召回后进行语义重排序。给定一个来查询 q_t，它被编码到一个共享密集嵌入空间中，并与文档块一起。然后我们选择与查询嵌入有最高余弦相似性的前 N 个候选者：

在随后的重新排序阶段，召回的候选块由一个更具表现力的相关模型重新评估，产生最终的前 k 个选择，并由 $\hat{\mathcal{T}}^{q_t}_{rerank}$ 表示。

为了支持对表格数据的准确推理，我们引入了一种在子查询推理涉及表格时才选择性调用的“程序-执行”机制。具体来说，我们检查在检索结果中是否有任何内容来源于表格。对于排在首位的集合 $\hat{\mathcal{T}}^{q_t}_{rerank}$ 中的每一个部分，我们应用映射函数（在公式 **??** 中）来提取其相关联的模式，从而生成一个表模式集合：

$$S^t = \{f(\hat{\mathcal{T}}_i) \mid \hat{\mathcal{T}}_i \in \hat{\mathcal{T}}^{q_t}_{rerank}\}. \quad (1)$$

。如果集合 S^t 为空，则跳过该模块。否则，我们将当前子查询 q_t 和对应的模式上下文作为输入，导出准确的答案。为此，我们利用关系数据的结构化查询执行，并使用 SQL 作为中间的形式化语言。一个专用工具 f_{SQL} 以 LLM 为后台生成可执行的 SQL 程序，并将其应用于预构建的 MySQL 数据库，形式化如下：

$$e_t = f_{SQL}(S^t, q_t). \quad (2)$$

对于子查询 q_t，TableRAG 可以从两种异构信息源中获益：SQL 数据库的执行结果 e_t 和文档数据库的文本检索结果 $\hat{\mathcal{T}}^{q_t}_{rerank}$。两种数据源均提供部分或完全的证据。它们引入了不同的失败模式：SQL 执行可能产生不正确的结果或执行错误，而文本检索可能导致上下文不完整或误导。因此，这些来源的结果可能相互

[1] https://github.com/mysql

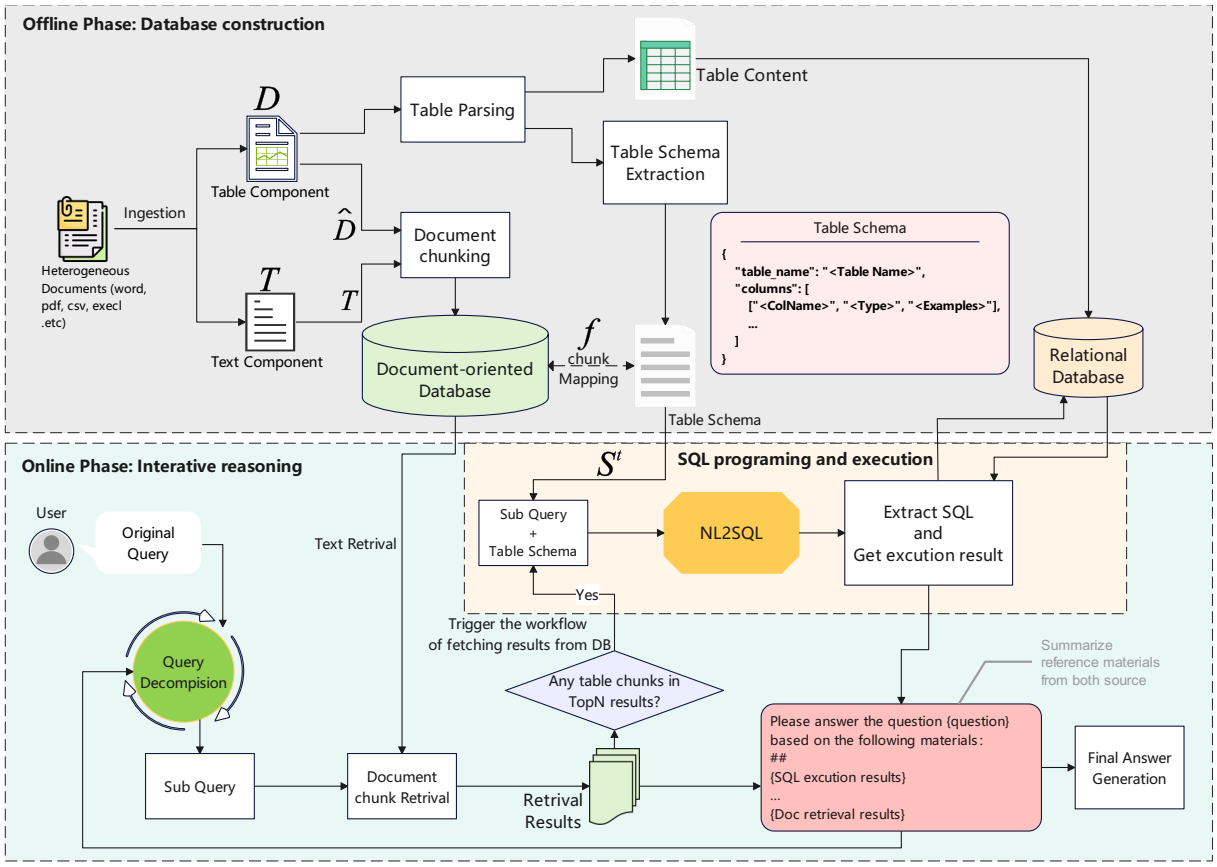

Figure 2: TableRAG 的整体架构。

补充或呈现矛盾。为了解决这一问题，我们采用一种组合推理机制。执行结果 e_t 和检索到的文本块 $\hat{T}^{q_t}_{rerank}$ 进行交叉检查，以验证一致性并指导答案选择。每个子查询的最终答案通过根据每个来源的证据效用来自适应地加权其可靠性来得出，$a_t = \mathcal{F}(e_t, \hat{T}^{q_t}_{rerank})$ 。

一旦查询分解模块确定不再需要进一步的子查询，TableRAG 便终止迭代推理过程，从而得到最终答案 $\mathcal{A} = a_T$ ，其中 T 表示执行的迭代总次数。

在本节中，我们介绍了 HeteQA，一个用于评估跨异构文档的多跳推理的新基准。

2.2 数据收集

HeteQA 需要高级操作，例如算术计算、嵌套逻辑等。为了平衡注释的准确性和可扩展性，我们采用了一种人机协作策略，将大型语言模型与人工验证相结合。构建流程分为三个阶段：

查询生成 我们从维基百科数据集中整理表格来源 (Chen et al., 2020) 。为了便于深入分析，我们将选择范围限制在至少有 20 行和 7 列的表格，并应用结构重复数据消除以消除相似模式中的冗余。对于每个保留下来的表格，我们定义了一套高级操作，例如条件筛选和统计聚合。这些操作充当构建复杂查询的原语。利用 Claude-3.7-sonnet [2] ，我们提示进行基于这些原语的查询合成。每个生成的查询都配有 SQL 和 Python 中的可执行代码。我们执行关联的代码并获得答案。最终的重复数据消除过程应用于查询和答案，促进数据集的多样性。完整的实施细节在附录 A 中提供。

为了确保正确性和可靠性，每个实例都由人工注释者进行人工检查。他们的任务是验证执行结果是否准确对应查询。在发现不一致的情况下，他们负责纠正底层代码和结果答案。

为了支持集成表格式和文本信息的查询，我们通过利用相关的维基百科文档来增强实例。具体而言，查询中的某些实体由人工注释者替换为基于参考的表述。例如，查询 “哪个司机 . . . ” 可以改写为 “司机 . . . 的国籍是什么”。这种实体替换可以修改问题的主体及其对应的答案，或者在保留原始答案的同时改变查询的措辞。注释指南和注释者简介详见附录 B 。

HeteQA 中的每个数据实例由一个查询、相应的答案、可执行的 SQL 语句和执行派生的答案组成。通过我们的数据收集管道，我们构建了 304 个高质量的例子，其中答案基于单一

[2] https://www.anthropic.com/claude/sonnet

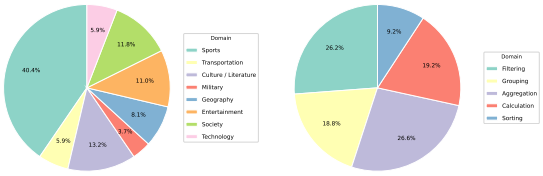

Figure 3: HeteQA 的领域分布和表格操作分布。

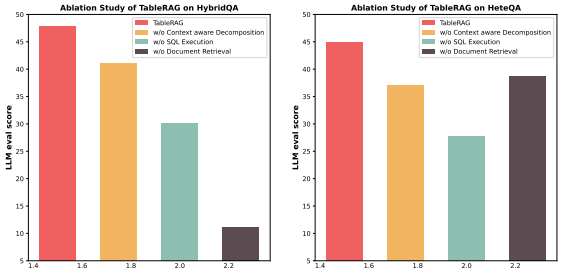

Figure 4: 基于 DeepSeek-V3 骨干网络的 HybridQA 和 HeteQA 基准上的消融研究。

来源 (82 %) 和多来源 (18 %)。最终的基准涵盖了 136 个不同的表和 5314 个维基知识实体。为了描述数据集，我们分析了它的语义域和表格推理操作类型。如图 3 所示，HeteQA 覆盖了 9 个语义上多样化的领域，并涵盖了 5 个主要类别的表格操作。总之，HeteQA 构成了一个结构多样且语义广泛的资源，用于推进对异构文档的问答研究。

3 实验

3.1 实验设置

我们在我们精心策划的 HeteQA 上评估 TableRAG 的性能，以及跨越两种设置的多个已建立的基准。

混合 QA (Chen et al., 2020) 一个涉及表格和文本信息的多跳问答数据集。对于我们的评估，我们仅保留包含超过 100 个单元格的表格的数据案例。

维基表格问题 (Pasupat and Liang, 2015b) 一个覆盖多样领域的 TableQA 数据集。这些查询需要进行多种数据操作，包括比较、聚合等。

在文本检索过程中，我们采用了 BGE-M3 系列模型。回调阶段中，我们保留前 30 个候选项，并通过重新排序选出其中的前 3 个。为了处理大规模输入，文本被分割成每段 1000 个标记的片段，相邻片段之间有 200 个标记的重叠。迭代循环的最大迭代次数限制为 5 次。对于基础大型语言模型（LLM），我们使用 Claude-3.5-Sonnet 作为代表性的闭源 LLM，而 Deepseek-V3、Deepseek-R1 和 Qwen-2.5-72B 则作为开源对应版本。在在线迭代推理过程中，为所有模块保持一致的后端。我们使用准确率作为评估指标，由 Qwen-2.5-72B 评估，结果为 0 或 1 的二进制分数。提示在附录中显示。

3.1.1 基线

我们通过将 TableRAG 与三种不同的基线方法进行对比评估其性能：(1) 使用大型语言模型直接生成答案。(2) NaiveRAG，它将表格数据作为线性化的 Markdown 格式文本处理，然后应用标准的 RAG 流程。(3) React (Yao et al., 2023)，这是一种基于提示的范式，用于在大型语言模型和外部知识源中协同推理和行动。(4) TableGPT2 (Su et al., 2024) 使用基于 Python 的执行模块在模拟环境中生成代码（例如，`Pandas`）以进行答案推导。这些基线方法的详细实现见附录 C 。

表 1 展示了不同大语言模型（LLM）骨干上的主要结果。有几个关键观察：(1) ReAct 框架在多源数据上展现出相对于普通 RAG 的优势，但在需要表格推理的单源数据上表现下降。这可以归因于多轮推理中的上下文不敏感性。被分解为多个子任务（如过滤或聚合）的查询因为信息不完整或错误传播而受到影响。(2) TableGPT2 仅在单源查询（如 WikiTQ）上产生可接受的结果，强调了其处理多源查询能力的有限性。这反映出在异质信息环境中的泛化能力不足。(3) TableRAG 超过所有基线，至少比最强的替代方案提高 10 %。值得注意的是，它在单源和多源数据上都表现出强劲的性能。这一性能提升归功于符号推理的结合，使其能够有效适应异质文档。此外，在不同 LLM 骨干上的性能一致性突显了 TableRAG 的架构通用性及其与广泛骨干兼容的能力。

为了阐明 TableRAG 框架中每个组件的相对重要性，我们将完整的架构与三个变体进行对比评估：(1) 没有上下文敏感的查询分解，在这种情况下，查询分解是在不依赖于检索到的表模式的情况下进行的。(2) 没有 SQL 执行，将 SQL 编程和执行模块替换为 markdown 表格格式。(3) 没有文本检索，仅通过基于表格的 SQL 执行操作，不利用诸如 Wikipedia 文档等文本资源。结果总结在图 4 中。所有模块都对 TableRAG 的整体性能做出了贡献，尽管它们在不同基准上的相对影响各异。在 HybridQA 上，文档检索显得尤为重要，因为它强调提取以实体为中心或数值的线索。相反，对于 HeteQA，SQL 执行更具影响力，因为查询涉及嵌套操作，得益于基于 SQL 的符号推理。这些发现突出了 TableRAG 文本检索和程序执行推理组件的互补设计。

我们通过检查 TableRAG 的执行迭代分布来

Method	Backbone	HybridQA	WikiTQ	HeteQA		
		-	-	Single-Source	Multi-Source	Overall
Direct	Claude-3.5	9.84	6.21	10.68	8.65	10.00
	DeepSeek-R1	24.42	12.20	3.40	13.46	6.77
	DeepSeek-V3	14.75	10.39	6.80	28.85	14.19
	Qwen-2.5-72b	11.47	7.37	4.85	12.50	7.42
NaiveRAG	Claude-3.5	20.28	82.60	33.20	40.35	34.54
	DeepSeek-V3	26.56	75.40	33.60	45.61	35.85
	Qwen-2.5-72b	22.62	66.33	23.07	36.84	25.66
ReAct	Claude-3.5	43.38	69.81	26.40	44.44	29.60
	DeepSeek-V3	38.36	63.40	21.14	47.39	26.07
	Qwen-2.5-72b	37.38	53.80	16.94	35.71	20.47
TableGPT2		9.51	63.40	35.60	16.67	32.24
TableRAG	Claude-3.5	47.87	84.62	44.94	40.74	44.19
	DeepSeek-V3	47.87	80.40	43.32	51.85	44.85
	Qwen-2.5-72b	48.52	78.00	37.65	43.96	38.82

Table 1: TableRAG 相较于基线模型在多个基准测试上的表现，以准确率测量。“多来源”表示需要表格和文本信息的问题，而“单一来源”指的是仅依赖一种信息源的问题。

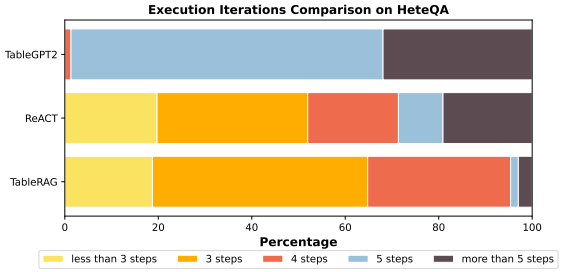

Figure 5: TableRAG、ReAct 和 TableGPT2 在 HeteQA 上的执行迭代比较。

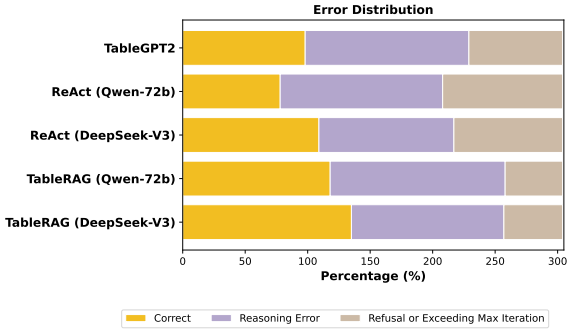

Figure 6: 基于 HeteQA 进行的 TableRAG、TableGPT2 和 ReAct 与 DeepSeek-V3 和 Qwen-2.5-72b 作为主干网络的误差分析。

评估其效率，如图 5 所示。执行长度分为四类：少于 3 步，3-5 步，正好 5 步，以及超过 5 步。在评估的方法中，TableGPT2 的平均执行步数最高，模态值集中在五步左右。相比之下，TableRAG 始终需要更少的步骤，在少于五步内解决了大约 63.55% 的实例，另外 30.00% 正好在五步内解决，只有极少数情况在给定的迭代限制下未解决。虽然 ReAct 在执行步骤上表现出类似的分布，但其整体性能显著低于 TableRAG。这些结果表明 TableRAG 在执行效率和推理准确性上均表现出色，这归功于其结合了基于 SQL 的表格推理。

我们在本节中对 TableRAG 进行了全面分析，其他结果见附录 **??** 。

除了评估 TableRAG 相对于既定基线的整体性能外，我们还进行了详细的错误分析，以表征预测失败的性质。大体而言，不正确的输出可以分为两个主要类别：（1）推理失败，可归因于 SQL 执行中的错误或中间查询分解的缺陷，以及（2）任务未完成，通常表现为拒绝回答或在超过最大迭代限制后终止。预测分布如图 6 所示。值得注意的是，TableGPT2 在这类失败中表现出最高频率，主要是因为它在整合来自维基文档的上下文线索方面能力有限。这个限制经常导致模型明确拒绝回应或承认其无能为力。相比之下，ReAct 缺乏上下文感知查询分解和代码执行模拟的机制，因此经常在本可以通过单一结构查询解决的问题上进行不必要的繁琐推理步骤。TableRAG 在所评估的方

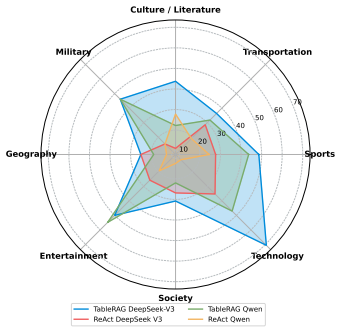

Figure 7: TableRAG 和 ReAct 在不同领域的性能分布。

法中显示出最低的失败率。其在五次迭代内持续产出有效响应的能力突出了其设计的有效性——特别是其使用上下文感知查询分解和选择性 SQL 执行规划的策略。

3.2 跨域预测

图 7 展示了 TableRAG 在各种主干 LLMs 实例化的情况下，与 ReAct 框架在不同领域的比较评估。结果显示，TableRAG 在大多数领域中稳定地优于 ReAct，证明了其在异构文档问答中的有效性。只有某些领域，例如文化领域，使用 Qwen 主干的 TableRAG 性能相对较弱。对数据分布的仔细检查表明，这种性能下降可能源于特定领域实例的稀疏性。

4 相关工作

4.1 检索增强生成

检索增强生成 (RAG) 已成为一种强有力的范式，用于缓解幻觉 (Zhang et al., 2023a) 并提高大型语言模型 (LLMs) 生成响应的可靠性 (Lewis et al., 2020; Guu et al., 2020) 。RAG 方法从知识库中检索，并随后将最相关的文档块纳入生成过程 (Gao et al., 2023; Zhu et al., 2024; Borgeaud et al., 2022) 。然而，这种简单的检索过程常常产生可能缺乏关键细节的噪声块，从而降低后续生成的质量。因此，最近的进展集中于任务自适应检索机制。在这方面的显著框架包括 Self-RAG (Asai et al., 2023) 、RQ-RAG (Chan et al., 2024) 等。尽管有这些创新，RAG 在处理异构背景时仍然面临挑战 (Satpute et al., 2024) 。

4.2 通过大型语言模型进行表格推理

表格推理指的是开发一个基于表格数据提供用户查询响应的系统 (Lu et al., 2025) 。表格推理的主流方法可以大致分为两类。第一类围绕通过提示工程利用大语言模型。例如，Tab-CoT (Jin and Lu, 2023) 应用链式思维（CoT）推理来建立一个表格结构化的推理过程。类似地，Chain-of-Table (Wang et al., 2024) 将 CoT 方法扩展到表格环境中，实现对更复杂表格查询的多步推理过程。第二类涉及使用程序来处理表格数据。Tabsqlify (Nahid and Rafiei, 2024) 采用文本到 SQL 的方法，将表格分解为较小的、具有上下文相关性的子表。DATER (Ye et al., 2023a) 采用少样本提示策略，将大型表格缩减为更易管理的子表，使用解析-执行-填充技术生成中间 SQL 查询。BINDER (Cheng et al., 2022; Zhang et al., 2023b) 整合了 Python 和 SQL 代码从表格中导出答案。InfiAgent-DABench (Hu et al., 2024) 利用一个基于大语言模型的代理，该代理进行规划、编写代码、与 Python 沙箱交互，并综合结果以解决基于表格的问题。

我们解决了现有 RAG 方法在处理结合文本和表格数据的异构文档时的局限性。目前的方法对表格的结构完整性造成损害，导致信息丢失和在全局、多跳推理任务中性能下降。为了解决这些问题，我们引入了一种名为 TableRAG 的 SQL 驱动框架，该框架将文本理解与精确的表格操作相结合。为了严格评估我们方法的能力，我们还提出了一个新的基准 HeteQA。对公共数据集和 HeteQA 的实验评估表明，TableRAG 显著优于现有的基线方法。

5

局限性 虽然 TableRAG 展现了强大的性能，但有几个限制值得关注：1. TableRAG 的效果与底层 LLMs 的能力密切相关。我们的实现利用了诸如 Claude、DeepSeek-v3 和 Qwen-72B-Instruct 等高容量模型，它们具有很强的泛化能力。缺乏专业指令微调的小型模型可能会表现出显著的性能下降。这表明，取得有竞争力的结果可能需要大量的计算资源。2. HeteQA 基准仅限于英语。这一限制源于跨多种语言策划高质量异质资源的难度。因此，跨语言泛化仍未被探索。在未来的工作中，我们计划将 HeteQA 扩展到多语言环境，从而扩大我们的评估框架的适用性和稳健性。

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from tril-

lions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. Rq-rag: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610*.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online. Association for Computational Linguistics.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2022. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Paulo Finardi, Leonardo Avila, Rodrigo Castaldoni, Pedro Gengo, Celio Larcher, Marcos Piau, Pablo Costa, and Vinicius Caridá. 2024. The chronicles of rag: The retriever, the chunk and the generator. *arXiv preprint arXiv:2401.07883*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. 2024. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*.

Ziqi Jin and Wei Lu. 2023. Tab-cot: Zero-shot tabular chain of thought. *arXiv preprint arXiv:2305.17812*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Weizheng Lu, Jing Zhang, Ju Fan, Zihao Fu, Yueguo Chen, and Xiaoyong Du. 2025. Large language model for table processing: A survey. *Frontiers of Computer Science*, 19(2):192350.

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. Tabsqlify: Enhancing reasoning capabilities of llms through table decomposition. *arXiv preprint arXiv:2404.10150*.

Panupong Pasupat and Percy Liang. 2015a. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Panupong Pasupat and Percy Liang. 2015b. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Ankit Satpute, Noah Gießing, André Greiner-Petter, Moritz Schubotz, Olaf Teschke, Akiko Aizawa, and Bela Gipp. 2024. Can llms master math? investigating large language models on math stack exchange. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pages 2316–2320.

Hesam Shahrokhi, Amirali Kaboli, Mahdi Ghorbani, and Amir Shaikhha. 2024. Pytond: Efficient python data science on the shoulders of databases. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 423–435. IEEE.

Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, et al. 2024. Tablegpt2: A large multimodal model with tabular data integration. *arXiv preprint arXiv:2411.02059*.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023a. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning. *arXiv preprint arXiv:2301.13808*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023b. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In

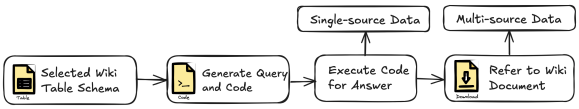

Figure 8: HeteQA 的数据集构建流程

Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '23, page 174–184, New York, NY, USA. Association for Computing Machinery.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023a. Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M Patel. 2023b. Reactable: Enhancing react for table question answering. *arXiv preprint arXiv:2310.00815*.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. *arXiv preprint arXiv:2105.07624*.

Yinghao Zhu, Changyu Ren, Shiyun Xie, Shukai Liu, Hangyuan Ji, Zixiang Wang, Tao Sun, Long He, Zhoujun Li, Xi Zhu, et al. 2024. Realm: Rag-driven enhancement of multimodal electronic health records analysis via large language models. *arXiv preprint arXiv:2402.07016*.

A HeteQA

A.1 提示设计

用于构建 HeteQA 基准的模板在表 **??** 中展示。

为了实现对表格结构的复杂推理，我们从基于维基百科的语料库中挑选了一部分广泛的表格。具体来说，我们仅保留了包含超过 20 行且至少有 7 列的表格。这个过滤过程将最初的 15,314 个表格缩减到符合结构丰富性标准的 1,345 个候选表格。为了进一步增强查询的多样性和减少数据集中的冗余性，我们应用了基于模式相似性的去重步骤。在多张表格共享相同列结构的情况下——例如，`1947 BAA draft` 和 `1949 BAA draft` 的条目——仅保留一个代表性实例。此程序产生了一个由 155 个独特表格组成的最终数据集。

如图 8 所示，每个数据实例是通过一个三级管道构建的，并通过 Claude 3.7 Sonnet API [3] 实现。

为了构建数据集，我们提示大型语言模型生成与查询相关联的 SQL 或 `pandas` 代码，并根据提供的表格模式进行生成。生成的代码随后针对表格执行以获取相应答案。不能产生可执行结果的实例将被弃置。为了促进数据集内部的多样性并避免冗余，我们消除任何显示重复查询或相同答案的实例。此过滤是通过正则表达式来检测词汇或语义重复执行的。此外，我们丢弃执行结果不确定的情况，例如当多个候选实体并列时的查询。经过自动化过滤后，所有保留的实例都经过人工验证过程。两位人工注释员独立评估每个查询、相应代码和结果答案。在出现不一致或错误的情况下，注释员会修改 SQL 或 Python 代码并纠正相关答案以确保准确性和连贯性。

[3] https://www.anthropic.com/claude/sonnet

B HeteQA 注释指南

B.1 标注者简介

两名标注者是母语为中文且拥有流利英语水平的志愿者。两位标注者都是专业的软件工程师，在 SQL 和 Python 编程方面有丰富的经验。

B.2 答案验证指南

目标

你的任务是根据相关的代码和表格，验证所提供的答案是否正确和完整地回答了自然语言查询。你还需要识别和处理模棱两可或不唯一的情况。

注释步骤

1. 验证答案

读取自然语言查询并检查附带的 Excel 表格、SQL 语句和 Python（pandas）代码。验证输出是否符合原始查询的语义和底层数据。为此，您可以对提供的 Excel 文件执行多次操作。

2. 修正任务（如有必要）

如果答案不正确，请同时修改代码和答案，以便它们正确地满足查询。确保修改后的代码是最小的、简洁的，并且逻辑上合理。

3. 歧义和平局情况

在查询没有产生唯一的解决方案时——无论是由于结果并列、条件不足，还是表达中的语义模糊——您可以采用正规的策略以确保有意义的处理：

- 选项 A: 修改查询以解决歧义（例如，进行澄清）。
- 选项 B：如果无法修复歧义或答案依赖于任意选择，则放弃该情况。

附加指南

保持查询、代码和答案之间的一致性。确保你的修改不会引入新模糊性或是假设，而这些是不存在于表格或查询中的。

B.3 文档引用指南

目标
您的任务是使用提供的维基百科实体和内容为原始查询添加一个额外的推理步骤。这将有助于将原始查询转化为一个更复杂的、多重推理的问题，需要更深入的推理。

输入数据
每个数据案例由以下内容组成：

- 原始查询
- 答案
- 与查询或答案相关的一组维基百科实体
- 每个实体对应的维基百科内容

任务概览
对于每个数据案例，检查原始答案是否在提供的 Wiki 实体中提到：
- 如果没有提及或模糊不清（例如，“Jordan” 既指篮球明星又指运动品牌），不要更改查询和答案。只需将情况标记为 “无需修改”。
- 如果答案实体存在于维基百科内容中，执行以下步骤：
1. 从其 Wiki 内容中识别关于答案实体的关键事实描述。
2. 向原始查询中添加一个推理步骤，通过这个关键事实得出答案。
3. 通过重写查询以合并这个额外的推理步骤并相应地更新答案来生成两个候选（查询，答案）对。

示例
原始查询：哪张专辑在 2013 年 Billboard 排行榜中名列第一？
原始答案：ArtPop
维基实体：ArtPop（专辑）
一个关键描述：这是由 Streamline 和 Interscope Records 于 2013 年 11 月 6 日发布的。
修改后的查询候选：
谁发行了在 2013 年 Billboard 排行榜上排名第一的专辑？答案：Streamline 和 Interscope 唱片公司

C 基线的实现细节

本节详细介绍了基线方法的实施步骤，以确保评估的公平性和可重复性。

对于 ReAct 框架，我们将表格数据预处理为 markdown 格式的纯文本。为了确保实验条件的一致性，我们采用与 TableRAG 模型相同的分块和检索配置。我们基于公开可用的

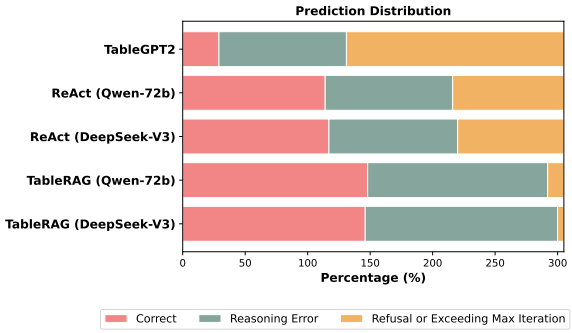

Figure 9: TableRAG、TableGPT2 和 ReAct 在 HybridQA 上的预测分布。

ReAct 实现通过一个包含 Thought 、Action 和 Observation 步骤的迭代推理循环来解决用户查询，最终通过 Finish 操作得到答案。最大迭代次数设置为 5，与 TableRAG 相同。

我们使用其正式发布的 TableGPT 代理实现[4] 来评估 TableGPT2。根据其 API 文档的说明，我们将文档内容和表格数据同时作为输入提供给 HumanMessage 类，以确保符合模型预期的使用方法：

```
，在空格处不换行，支持自动换行，说明位置在底部，保留空格，行号在左侧，数字与正文间隔 5pt，不显示空格，不显示字符串中的空格，不显示制表符，制表符宽度为 2
from typing import TypedDict
from langchain_core.messages import HumanMessage

class Attachment(TypedDict):
    """Contains at least one dictionary with the key f
    filename: str

attachment_msg = HumanMessage(
    content="",
    # Please make sure your iPython kernel can access
    additional_kwargs={"attachments": [Attachment(file
)
```

我们对所提出的基准 HETEQA 和公开可用的 HYBRIDQA 数据集进行了系列辅助分析。这些分析提供了超越主要评估指标的进一步关于 TableRAG 的行为和局限性的见解。

C.1 混合问答分析

我们通过检查数据域上的性能分布并进行详细错误分析，将调查扩展到 HYBRIDQA 数据集，以 DEEPSEEK-V3 作为骨干模型。结果总结在图 10 和图 9 中，揭示了与在 HETEQA 上观察到的模式相一致的模式。这种并行进一步验证了我们在 HeteQA 设置中观察结果的普遍性。

为了阐明顶级 k 检索参数对 TableRAG 性能的影响，我们进行了系统的敏感性分析。虽然我们的主要实验设置固定了 $k = 3$ ，但我们在

[4] https://github.com/tablegpt/tablegpt-agent

table	List_of_Australian_films_of_2012_0
query	Who wrote and starred the comedy film released in the second half of 2012 (July-December) that had the highest number of cast members in the List of Australian films of 2012?
sql_query	Which comedy film released in the second half of 2012 (July-December) had the highest number of cast members in the List of Australian films of 2012?
sql	
sql_ans	Kath & Kimderella
answer	Riley, Turner, and Magda Szubanski

```sql
SELECT
  title,
  LENGTH(cast_subject_of_documentary) - LENGTH(
    REPLACE(
      cast_subject_of_documentary, ',',
      ''
    )
  ) + 1 AS cast_count
FROM
  `list_of_australian_films_of_2012_0_sheet1`
WHERE
  genre LIKE '%Comedy%'
  AND (
    release_date LIKE '%July%'
    OR release_date LIKE '%August%'
    OR release_date LIKE '%September%'
    OR release_date LIKE '%October%'
    OR release_date LIKE '%November%'
    OR release_date LIKE '\ufffdcember%'
  )
ORDER BY
  cast_count DESC
LIMIT
  1;
```

Table 2: HeteQA 的一个例子。

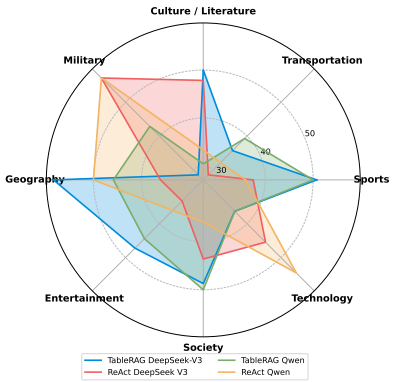

Figure 10: 不同领域在 HybridQA 上的模型表现。

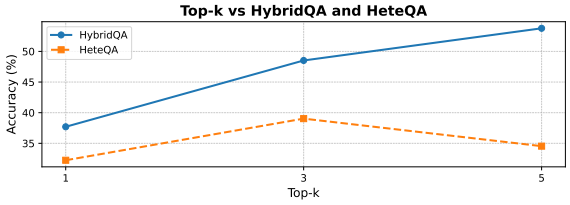

Figure 11: 对 HeteQA 上 TableRAG 的超参数 k 分析。

HeteQA 数据集上拓展了调查，使用 DeepSeek-V3 作为检索骨干，并在集合 $1, 3, 5$ 中变化了 k 。所得的性能指标在图 11 中进行了说明。

我们的观察揭示了在基准测试中对于 k 选择的不同行为。值得注意的是，HybridQA 在较高的 k 值时表现优异。这种效果很可能是由于其表中缺乏去重性，导致从多个表或维基百科文档中获取的重叠或相似信息在检索过程中起到了建设性作用。总体而言，设置 $k = 3$ 达到了有效的平衡，实现了在维持计算效率的同时在两个基准上都有稳健的性能。这种效率的提升源于 top- k 检索大小与 LLM 后续上下文长度之间的直接关系，强调了此超参数在优化准确性与资源消耗之间权衡的实际重要性。

D 检查清单

有害信息和隐私 我们提出了一种新的 RAG 解决方案来解决与异构数据相关的多跳问题，而不涉及任何有害信息或隐私。此外，我们提供一个包含来源于维基百科的表格和文本的异构基准。所有数据都是公开可用的，不含个人信息，也不涉及有害内容。

许可证和意图 我们在此提供我们使用的许可证：

- Claude 3.5 Sonnet (https://www.anthropic.com/legal/aup)
- Qwen2.5-72B-Instruct (https://huggingface.co/Qwen/Qwen2.5-72B-Instruct/blob/main/LICENSE)
- DeepSeek-V3 (https://huggingface.co/deepseek-ai/DeepSeek-V3/blob/main/LICENSE-MODEL)
- TableGPT Agent (Apache License 2.0) 和 TableGPT2-7B (https://huggingface.co/tablegpt/TableGPT2-7B/blob/main/LICENSE)

我们对这些现有材料的使用与它们的预期用途一致。

我们提出了一种新颖的检索增强生成（Retrieval-Augmented Generation, RAG）框架，该框架将传统的文档检索与通过 SQL 进行的结构化数据查询相结合，旨在提升与表格相关的问题回答任务的性能。该框架包括两个基本阶段：离线数据库构建和在线互动推理。与传统的以文档为中心的 RAG 方法相比，我们的架构提供了额外可靠的 SQL 执行结果——当表格片段参与检索过程时。这一补充来源缓解了生成模型在处理结构化表格数据时的局限性。

为了更全面地评估我们的框架，我们进一步构建了一个名为 HeteQA 的异构基准。这个基准旨在评估处理跨异构文档的多跳推理任务的能力。基准实例最初由大型语言模型生成，然后由经验丰富的程序员和数据库管理员进行严格验证，以确保其正确性和真实性。

E 提示 。

本章节展示了 TableRAG 的提示。

Prompt Template for SQL generation

Multi-Hop Table Reasoning Query Generator

Task
Generate a genuine multi-hop reasoning query based on the provided markdown table, along with SQL and pandas solutions in a structured JSON format.

Input
A markdown formatted table schema.

Output Requirements
Provide exactly ONE multi-hop reasoning query that:
- Requires sequential analytical operations where each step depends on the previous result
- Cannot be broken down into separate independent questions
- Is solvable using both SQL and pandas
- Is relevant to the data domain in the table
- Mention the table name in query to indicate the source table file

Operations to Consider in Your Sequential Reasoning Chain
- Group or Aggregate
- Filtering subsets
- Calculating percentages or ratios between groups
- Comparing specific subgroups
- Rank or order
- Finding extremes (max/min) of aggregated values
- Computing difference or sum
- Finding correlations

Example Operations Combinations
- Filter Group Rank in groups
- Group Sum Compare
- Filter Calculate percentage Rank
- Group Aggregate Filter on aggregate

Guidance for True Multi-Hop Queries
A proper multi-hop query requires sequential operations where each step builds on the result of the previous step. For example:

GOOD (True multi-hop): "What was the average lap time among the top 5 ranked drivers in the team which had the best average lap time?"
- This requires first finding the team with best average lap time
- Then identifying the top 5 drivers in that specific team
- Finally calculating the average lap time of just those drivers

BAD (Separable questions): "Which team had the best average lap time, and what was that average among the top 5 ranked drivers?"
- This could be answered as two separate questions

Format your response as a single JSON object with this structure:
{
"query": "Clear natural language question requiring true multi-hop reasoning",
"operations_used": ["List operations used, such as: filtering, aggregation, grouping, sorting, etc."],
"sql_solution": "Complete executable SQL query that solves the question",
"pandas_solution": "Complete executable pandas code that solves the question",
"result_type: "The type of the result, must be either number or entity."
}
Ensure your query truly requires chained reasoning where later steps must use results from earlier steps and generates one answer.

Prompt Template for Query Decomposition ???

```
tools = [{
    "type": "function",
    "function": {
        "name": "solve_subquery",
        "description": "Return answer for the decomposed subquery",
        "parameters": {
            "type": "object",
            "properties": {
                "subquery": {
                    "type": "string",
                    "description": "The subquery to be solved"
                }
            },
            "required": [
                "subquery"
            ],
            "additionalProperties": False
        },
        "strict": True
    }
}]
```

Next, You will complete a table-related question answering task. Based on the provided materials such as the table content (in Markdown format), you need to analyze the Question. And try to decide whether the Question should be broken down into subquerys. After you have collected sufficient information, you need to generate comprehensive answers.
You have a "solve_subquery" tool that can execute SQL-like operations on the table data. It accepts natural language questions as input.

Instructions:
1. Carefully analyze the user query through step-by-step reasoning.
2. If the query requires multiple pieces of information, more than the given table content:
- Decompose the query into subqueries
- Process one subquery at a time
- Use "solve_subquery" tool to retrieve answers for each subquery
3. If a query can be answered by table content, do not decompose it. And directly put the origin query into the "solve_subquery" tool.
The "solve_subquery" tool can solve complex subquery on table via one tool call.
4. Generate exactly ONE subquery at a time.
5. Write out all terms completely - avoid using abbreviations.
6. When you have sufficient information, provide the final answer in this format:
<Answer>: [your complete response]

Table Content: { table_content }
Question: { query }
Please start!

Prompt Template for Intermediate Answer Reasoning

You are about to complete a table-based question answering task using the following two types of reference materials:

Content 1: Original content (table content is provided in Markdown format)
{ docs }

Content 2: NL2SQL related information and SQL execution results in the database
the user given table schema
{ schema }

SQL generated based on the schema and the user question:
{ nl2sql_model_response }

SQL execution results
{ sql_execute_result }

Please answer the user's question based on the materials above.
User question: { query }

Note:
1. The markdown table content in Content 1 may be not complete.
2. You should cross-validate the given two materials:
- if the answers are same, you may directly output the answer.
- If the SQL shows error, such as "SQL execution results", try to answer solely based on Content 1.
- If the two material shows conflict, carefully evaluate both sources, explain the discrepancy, and provide your best assessment.

Prompt Template for Answer Evaluation

We would like to request your feedback on the performance of the AI assistant in response to the user question displayed above according to the gold answer. Please use the following listed aspects and their descriptions as evaluation criteria:
- Accuracy and Hallucinations: The assistant's answer is semantically consistent with the gold answer; The numerical value and order need to be accurate, and there should be no hallucinations.
- Completeness: Referring to the reference answers, the assistant's answer should contain all the key points needed to answer the user's question; further elaboration on these key points can be omitted.
Please rate whether this answer is suitable for the question. Please note that the gold answer can be considered as a correct answer to the question.

The assistant receives an overall score on a scale of 0 OR 1, where 0 means wrong and 1 means correct.
Dirctly output a line indicating the score of the Assistant.

PLEASE OUTPUT WITH THE FOLLOWING FORMAT, WHERE THE SCORE IS 0 OR 1 BY STRICTLY FOLLOWING THIS FORMAT: "[[score]]", FOR EXAMPLE "Rating: [[1]]":
<start output>
Rating: [[score]]
<end output>

[Question]
question

[Gold Answer]
golden

[The Start of Assistant's Predicted Answer]
{ gen }