分解驱动的多表检索与推理：数值问答

Feng Luo[1†]、Hai Lan[2], Hui Luo[3], Zhifeng Bao[2], Xiaoli Wang[4], J.Shane Culpepper[2], Shazia Sadiq[2]RMIT大学 [2]昆士兰大学 [3]伍伦贡大学 [4]厦门大学 feng.luo@student.rmit.edu.au, h.lan@uq.edu.au, huil@uow.edu.au, zhifeng.bao@uq.edu.au, xlwang@xmu.edu.cn, s.culpepper@uq.edu.au, shazia@eecs.uq.edu.au

arXiv:2603.07950v1 [cs.DB] 9 Mar 2026

摘要—**本文研究大规模表集合（例如在线数据存储库）上的数值型多表问答（多表问答）问题。这项任务对于许多分析应用至关重要。现有的多表问答解决方案（如文本到SQL或开放域多表问答方法）是为数据库设计的，应用于大规模表集合时表现不佳。其主要局限包括：（1）对复杂表关系的支持有限；（2）无法有效地大规模检索相关表；（3）答案生成不准确。为克服这些局限，我们提出了 DMRAL，即面向大规模表集合多表问答的分解驱动的多表检索与回答框架，该框架包括：（1）构建表关系图以捕捉表之间的复杂关系；（2）表格对齐问题分解器和覆盖率感知检索器，通过提升问题分解质量并最大化检索到的表的问题覆盖率，共同实现从大规模语料库中有效识别相关表；（3）子问题引导推理器，通过根据子问题逐步生成和优化推理程序，从而得出正确答案。在两个多表问答数据集上的实验表明，DMRAL 显著优于现有最先进的多表问答方法，在表格检索和答案准确率方面的平均提升分别达到 24% 和 55%**。

索引词—**多表问答，大语言模型**

一、引言

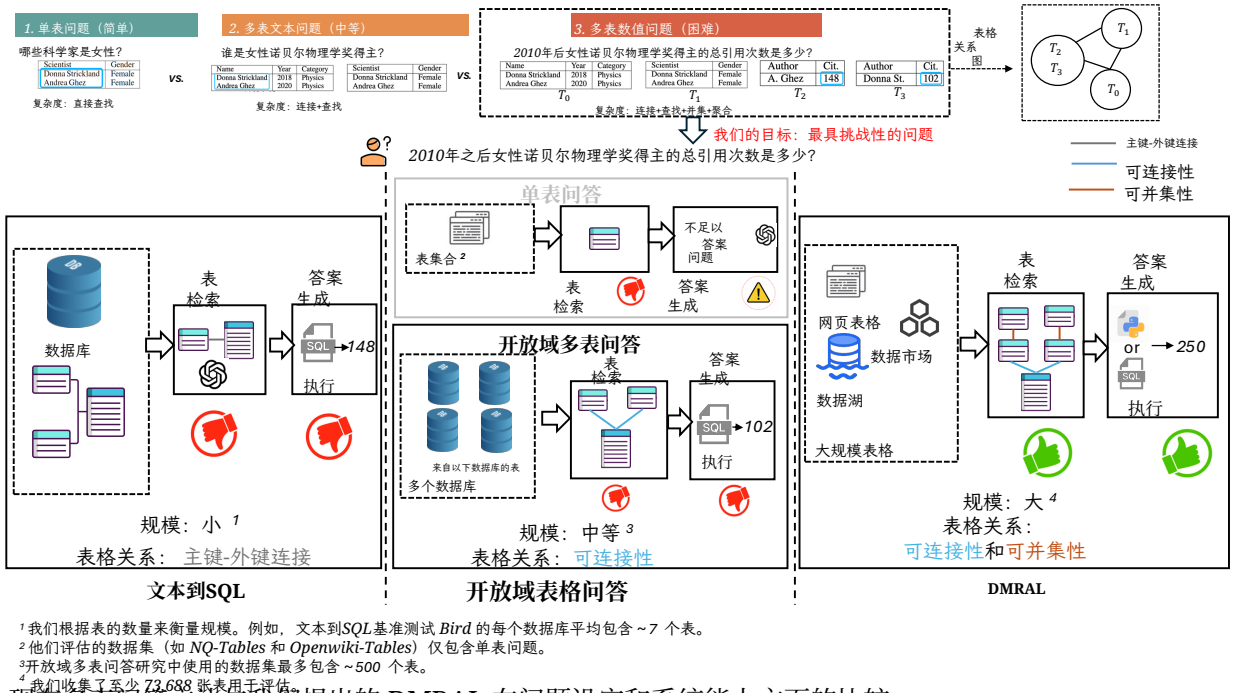
多表问答（MTQA）是一项广为人知的任务，要求从多个表中识别并整合信息，以回答一个问题 [1]。在这些问题中，需要数值答案 的问题比需要文本答案的问题复杂得多。实证证据表明，两者的性能差距显著：即使使用标准答案表，数值答案的准确率也约为 55%，而文本答案则达到 88% [2], [3],，这凸显了多表问答系统准确支持数值问题的必要性 [4]。与此同时，大规模表集合日益普及——例如网络上的表、公共数据湖中的表，或通过数据市场获取的表——为多表问答带来了丰富的机遇。在这些场景中，用于回答数值问题的宝贵数据往往分散在这些杂乱的表中。尽管前景广阔，但处理此类表也面临独特挑战，因为它们规模庞大（例如，数万张表）、元数据不完整（例如，缺少列标题） [5], [6], ，且表间关系复杂，具体而言，包括可并集性（可根据相似列标题进行并集的表）和可连接性（可根据匹配列进行连接的表） [7], [8]。本文研究大规模表集合上的数值型多表问答。这一场景颇具挑战性，但随着真实世界数据获取和使用趋势的发展，也日益贴近现实。

当前的多表问答方法大致分为两类：文本到*SQL* 和开放域表格问答。如图 1 所示，文本转SQL方法 [9], [10] 最初是为单个关系数据库开发的，通常涉及不到十张表，并依赖定义良好的数据库模式（即完整的表元数据和主键-外键约束[11]）。我们的问题中不具备这类模式信息，规模也有限，因此文本转SQL方法并不直接适用，我们将在第 II-B 节简要说明这一点。开放域表格问答又分为 **(i)** 单表问答 [12], [13],，它假定答案所需的信息仅来自单张表，因此与我们的场景不兼容；以及 **(ii)** 开放域多表问答 [1], [14], ，它结合来自多张表的证据。虽然开放域多表问答与我们的问题更为接近，但其设计目标是规模适中的语料库（即数百张表），这些语料库由多个关系数据库汇集而成。这类方法通常首先利用大语言模型的内在知识将问题分解为一组子问题，然后根据子问题与表之间的相关性得分，并利用表可连接性检索相关表，最后调用大语言模型在检索到的表上生成 SQL。

然而，将现有的开放域多表问答方法应用于大规模表集合时，会面临三大主要局限。**(L1)**对复杂表关系的支持有限：开放域多表问答忽视了可并集性等复杂表关系 [15],，这限制了其处理此类表语料库时的鲁棒性。**(L2)** 无法有效地大规模检索相关表：开放域多表问答依赖大语言模型分解子问题以检索表，而低质量分解产生的错误可能传播至检索环节，导致检索效果不佳。**(L3)**答案生成不准确：生成 SQL 或 Python 等程序并运行程序以推导答案，是回答数值问题的一种有前景的方法，但现有方法很少能生成完全正确的程序（例如，包含不正确的连接），因此答案准确率较低 [16]。

我们的贡献。为了解决这些局限性，我们提出了

†本研究是在访问昆士兰大学期间完成的。



[1] 我们根据表的数量来衡量规模。例如，文本到SQL基准测试 *Bird* 的每个数据库平均包含 ~7 个表。
[2] 他们评估的数据集（如 *NQ-Tables* 和 *Openwiki-Tables*）仅包含单表问题。
[3] 开放域多表问答研究中使用的数据集最多包含 ~500 个表。
[4] 我们收集了至少 73,688 张表用于评估。

图1：现有多表问答方法与我们提出的 DMRAL 在问题设定和系统能力方面的比较。

DMRAL，这是一种面向大规模表集合中的多表问答（MTQA）的分解驱动的多表检索与回答框架。我们的主要贡献如下：

- 我们开发了一套预处理 流程（§II-C），用于构建表关系图，以捕捉复杂的表格关系（即解决**L1**）。
- 我们提出了一种有效的、由分解驱动的多表检索策略，以解决**L2**，该策略包括：（1）表格对齐的问题分解器（§III），通过识别不同的信息需求并将其与底层表格结构对齐，提升分解质量；（2）覆盖感知检索器（§IV），通过覆盖度评分和验证机制，最大限度地覆盖问题所需信息，从而有效地大规模检索相关表，并缓解级联检索错误。
- 我们提出了一种子问题引导推理器（§V），通过引导大语言模型基于分解后的子问题逐步生成并优化程序，从而提高答案准确率（即解决**L3**）。
- 为全面评估这项任务，我们准备了两个大规模数据集 SpiderWild 和 BirdWild，它们包含从真实世界表格来源中整理的 73,688 个和 109,949 个表 [17]–[19]（§VI）。大量实验表明，DMRAL 有效性显著，在识别相关表方面平均提升 24%，在生成准确答案方面平均提升 55%（§VII）。

总之，DMRAL 是一个稳健、可扩展且可追溯的框架，专为大规模表集合上的数值型多表问答而设计。DMRAL 支持细粒度的追踪与验证（例如，检索到的表是否合适、推理程序是否正确，以及子问题是否拆解得当）。这种可追溯性至关重要，因为它使答案的推导过程清晰透明，并提供可操作的见解，以便诊断和改进各个组件。

II. 问题、相关研究与解决方案概述

A. 问题定义

一个表 T由一组列 $T_c = \{c_1, \ldots, c_m\}$和行 T_r以及相关元数据组成。元数据包括表标题 T_p和列标题 $T_h = \{h_1, \ldots, h_m\}$，它们可能部分缺失。我们将表集合定义为 $\mathcal{T} = \{T^1, T^2, \ldots, T^{|\mathcal{T}|}\}$。这些表通常通过两种具有代表性的表间关系相互关联：

- 可连接性：</style}若表 T^i和 T^j中至少存在一列分别来自 T_c^i和 T_c^j，且这两列包含重叠或语义相似的值，则认为这两个表可连接。
- **可并集性：** 若表 T^i和 T^j的列标题 T_h^i和 T_h^j足够相似，能够在两个表之间进行一对一对齐，则认为这两个表可并集。

定义 *1*（数值型多表问答）：给定一个表集合 $\mathcal{T}$和一个需要数值答案的数值问题 q，其目标是通过学习函数 f来计算正确答案 ans：$(\mathcal{T}, q) \rightarrow ans$，其中答案 ans 通过识别相关表的一个子集 $\mathcal{T}_q \subseteq \mathcal{T}$并在其上进行推理 $\mathcal{T}_q$得出。

B. 相关工作

文本到SQL。文本到SQL是一项由来已久的任务，旨在将自然语言问题转换为可在关系数据库上执行的SQL查询。现有的文本到SQL方法分为三种范式：基于规则、神经方法和基于大语言模型的方法。基于规则的方法使用手工编写的语法和语义解析器 [20]–[22]，但需要大量人工

工程投入，且难以泛化到新领域 [23]。神经方法利用深度模型 [24]–[26] 来提升模式理解能力，但仍受模型容量和高质量训练数据可用性的限制。基于大语言模型的方法通过提示工程 [27], [28],、监督微调 [29]，以及工具增强型/智能体推理 [9],[30]–[32]来生成SQL。例如，CHESS [32]提出了一种多智能体流水线，分阶段进行模式选择、SQL生成和验证。OpenSearch-SQL [28] 提出了一种包含四个阶段的多智能体协作方法：预处理、提取、生成和优化，并取得了最先进的结果 [23]。

尽管这些方法取得了成功，但由于两个关键局限，它们无法直接应用于大规模表集合上的多表问答。首先，它们假设通过主键-外键连接 [33] 存在显式的表关系，并且需要完整元数据来进行模式链接 [34], ，而这两者在真实世界的表集合中均不可用。其次，最先进的基于大语言模型的方法受限于大语言模型的上下文长度，无法扩展到大规模表语料库 [35]。鉴于这些局限，我们不将文本转SQL方法作为主要基线方法，但提供了两项互补比较：（i）适配评估，即通过数据库物化在相同表集合上运行文本到SQL；（ii）在其原始设置下进行的二次评估（§VII-E）。

开放域多表问答。开放域多表问答在表集合上运行，首先利用分解后的子问题检索相关表，然后使用大语言模型直接生成SQL查询 [36]。为识别相关表，MMQA [1] 将该问题表述为多跳检索问题。相比之下，JAR [14] 通过联合考虑子问题–表相关性与整体表相关性，将该问题表述为优化问题。然而，这两种方法都是为数据库表设计的，这限制了它们识别可合并表的能力。此外，MMQA容易受到早期检索错误所引入的级联错误的影响 [37], ，从而降低其有效性。相反，JAR因所采用的优化算法而承受较高的计算开销，限制了效率。

其他较为相关的研究。开放域单表问答 [12], [13], [38] 旨在仅使用
从多样化的表集合中检索最相关的表，依赖于专门的单表检索方案，例如 [39] ，而不考虑表间关系。这一基本假设使其无法整合多个表中的信息来回答问题。

C. 解决方案概述

DMRAL 首先通过一个预处理 流程，构建表关系图 $\mathcal{G} = (\mathcal{V}, \mathcal{E})$，以高效捕捉复杂表关系。在该图中，每个节点$v \in \mathcal{V}$ 代表一个由 [40]中的可并集性识别出的可合并表簇。如果簇中的任意一对表可连接，则边 $e \in \mathcal{E}$ 会连接这些簇，可连接性根据 [14]中的定义确定。基于这个预处理后的图（具体示例见图 1），DMRAL 通过三个核心模块回答问题，如图2所示：

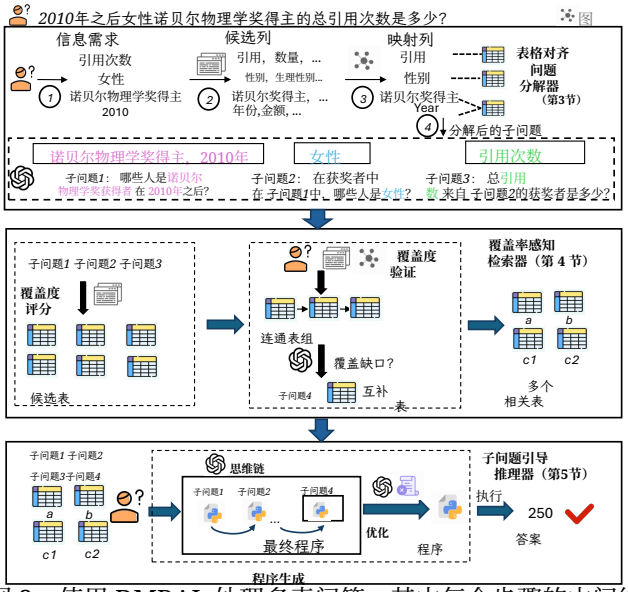


图 2：使用 DMRAL 处理多表问答，其中每个步骤的中间结果均可追踪和优化。

表格对齐问题分解器(§III)将输入问题分解为多个子问题。为提升分解质量，它提出了一种四步分解方法：(1) 从问题中提取信息需求，(2) 将每个信息需求与列对齐，(3) 选择信息需求与列之间最有前景的映射，(4) 使用基于大语言模型的分解，将语境上连贯的信息需求分组并生成子问题。

覆盖感知检索器(§IV) 基于分解后的子问题检索多个相关表。为支持对大规模表集合的有效检索，该方法首先引入一种基于学习的覆盖度评分函数，为每个子问题检索候选表。随后，它提出一种覆盖度验证函数，先构建能够共同回答问题的连通表组，再识别互补表以弥补任何潜在的覆盖缺口。

子问题引导推理器(§V) 在检索到的表上生成可执行程序（例如 SQL 或 Python），以推导出最终答案。为提高答案的可靠性，它首先使用思维链提示，依据一系列分解后的子问题逐步生成中间程序。随后，它采用执行引导的细化机制，对生成的程序进行验证并迭代修订。

III. 表格对齐问题分解器

A. 有效分解的原则

对复杂用户问题进行正确分解对于多表问答至关重要，因为它能够实现精准检索 [14]。给定一个问题 q，我们将其中包含的信息需求定义为一个集合 $\mathcal{I}(q)$，其中每个元素 $I_q \in \mathcal{I}(q)$代表 q 中包含的特定概念、实体或条件，而这些内容是推导出正确答案所必需的。

我们认为，高质量的问题分解应遵循以下三个指导原则：（1）**完整性**：生成的子问题必须共同涵盖

q中包含的所有信息需求。遗漏问题的任何方面，都可能导致答案不完整，无法充分满足用户意图；（2）**非冗余性**：各子问题的信息需求之间不应有重叠。冗余信息会带来不必要的计算，增加检索延迟，还可能为推理提供误导性输入；（3）**表特异性**：每个子问题都针对单个表或一组可合并表。这样可以降低多表连接的复杂性，并确保每个子问题都能在连贯的上下文中得到解答。

尽管大语言模型已展现出进行分解的潜力 [1],[41], [42] ，并且与以往工作相比提高了分解的准确率 [43], [44],，但我们的实证分析（§VII-B）表明，仅使用大语言模型进行分解往往无法满足上述原则，从而导致错误的答案。

B. 从问题到表特定子问题

为了解决这些局限并满足上述原则，我们提出了一种四步分解方法，使子问题始终与表集合的结构信息保持一致。

1) 识别信息需求：为了从每个问题中提取最显著的信息需求，我们首先使用斯坦福 Stanza 工具包将问题解析为成分树 [45]。然后，我们提取所有名词短语，这些短语表示所需的核心概念和实体；同时，根据句法标签 [46], 纳入可能表示条件的形容词短语和动词短语。这将生成一组全面的“信息需求”，以确保**完整性**。

2）混合列匹配：为确保信息需求与列相匹配，我们通过拼接表标题 T_p、列标题 h_j和来自 c_j 的不同单元格值（以空格分隔），构造文本片段来表示表 T 中的每一列 c_j 。由于信息需求在词汇和语义上都可能与表内容有所不同，我们采用 M3-Embedding（M3） [47], ，这是一种统一的嵌入模型，旨在对文本的词汇与语义特征进行编码。随后，我们使用 M3 对表集合$\mathcal{T}$ 中的所有列片段进行编码，并用 FAISS [48]建立索引，以支持高效的相似度搜索。对于每项信息需求 I_q，我们对其进行 M3 嵌入编码，并根据相似度分数检索排名前 30 的列片段。经经验评估，我们选择 30 作为检索深度，以平衡相关性与噪声（不相关的列）。

3）上下文感知列消歧：对于每项信息需求 I_q，我们找出候选列 $\mathcal{C}_{I_q}$及其相似度分数 $\text{sim}(I_q, c_{I_q})$，其中每个 $c_{I_q} \in \mathcal{C}_{I_q}$ 均对应一个分数。为指导子问题生成，我们选取信息需求与列之间最有希望的映射，以确保各项信息需求之间的上下文连贯性。

由于问题 q 的信息需求可能彼此相关，因此它们应对应于同一张表或一组可连接表中的列。为此，我们使用包含任何所需上下文知识的表关系图 $\mathcal{G}$ ，并定义“上下文相关性分数”，以便为每项信息需求联合评估所有候选列。上下文相关性分数。给定一个特定映射 M，该映射为每个 $I_q \in \mathcal{I}_q$ 从 $\mathcal{C}_{I_q}$ 分配一列给 I_q ，上下文相关性分数 $R(M, \mathcal{I}_q)$ 定义为 $\sum_{(I_q, c_{I_q}) \in M} \text{sim}(I_q, c_{I_q})$，若 $\mathbb{I}\left(\{T(c_{I_q}) \mid I_q \in \mathcal{I}(q)\}, \mathcal{G}\right) = 1$。否则， $R(M, \mathcal{I}_q) = 0$。此处， $T(c_{I_q})$ 表示包含列 c_{I_q} 的表，而 $\mathbb{I}(\cdot, \mathcal{G})$ 是一个指示函数：若所有选定的表都属于 $\mathcal{G}$ 中构成一个连通分量的节点，则返回 1。

我们的目标是确定一个映射 M ，使其在所有可能的映射中获得最高的上下文相关性分数。然而，候选映射的数量会随着信息需求数量的增加而呈指数级增长，而信息需求的数量可能很大。因此，穷举搜索在计算上效率不高。为此，我们对 $\mathcal{G}$采用贪心策略。该策略包含以下步骤。

步骤 *1*：对信息需求排序。我们根据每项信息需求与其候选列之间的最高相似度分数，按降序排列所有信息需求。

步骤 *2*：初始化映射。我们通过为排名最高的信息需求选择相似度分数最高的候选列，来初始化映射 M。

步骤 *3*：逐步扩展映射。我们依次遍历其余已排序的信息需求。对于每项信息需求，我们选择一个候选列，该候选列满足以下条件：(1) 属于与当前映射 M中已包含的表在 $\mathcal{G}$ 中同一连通分量内的表；(2) 在这些连通候选项中具有最高相似度分数。如果没有候选项满足这些要求，则认为当前扩展失败。在这种情况下，我们回溯到步骤 2，根据相似度分数选择次优候选列，重新尝试该过程。

4）问题分解：获得最优列选择后，我们根据所选列所属的表对信息需求进行分组。随后将这些与表对齐的分组发送给大语言模型，为每组生成一个子问题。这一步通过确保每个子问题对应一组互不重叠的目标需求来**减少冗余**，并通过限定每个子问题的范围，使**表特异性**更加一致。

IV. 覆盖感知检索器

检索出完整且相关的表集是准确进行多表问答的关键要求。然而，现有方法要么计算成本高昂 [14] ，要么采用过于僵化的多跳流水线，导致错误率上升 [1]。为此，我们提出覆盖感知检索器，以支持更高效的多跳检索。该方法依赖两项关键创新：（1）一种覆盖度评分函数，优先考虑问题语义覆盖，以提升早期阶段候选选择的有效性（§IV-A）；（2）一种覆盖度验证函数，利用剩余子问题检测并纠正任何信息缺失（§IV-B）。

A. 通过基于学习的评分最大化覆盖率

我们的方法采用两阶段流水线。首先，我们使用FAISS 和 M3表嵌入执行候选检索。然后，我们应用更精细的重排序模型，以提高精确率。

粗粒度检索。我们将 $\mathcal{G}$ 中的每个表簇表示为一个文档，方法是拼接表元数据并使用M3对其进行编码。生成的嵌入随后使用FAISS建立索引。我们避免嵌入全部内容，因为将全部内容展平可能会超出大型表（例如包含 $>$ 1K行的表）的编码器上下文限制，并且会因高频实体（例如年份）引入虚假匹配。对于每个子问题sq_i，我们通过查询索引检索所有候选簇。

学习式评分。 粗粒度检索常常会因表面的语义相似性引入不相关文档，从而在后续阶段传播错误。为了在候选选择过程中提高精确率，我们训练一个评分函数 $f_\theta(q, T_q)$，用于估计候选表 T_q 相对于问题 q的语义覆盖度 。我们使用 ColBERTv2 [49] 来创建 f_θ ，因为它在捕捉上下文交互方面具有有效性。 T_q使用包含紧凑表元数据的同一文档进行表示，以便在ColBERTv2的上下文限制下实现可扩展性。

为了构建训练数据，我们纳入单表问答数据集 [17], [18]。每个训练实例由一个问题 q、一个正例表 T^+ （即包含正确答案的真实表）以及一个困难负例表 T^-组成；困难负例表是通过从 T^+ 中移除含答案列而创建的，以模拟部分相关性。我们使用以下基于间隔的排序损失训练 f_θ ：

$$\mathcal{L} = \sum_{(q,T^+,T^-)} \max(0, 1 - f_\theta(q, T^+) + f_\theta(q, T^-)).$$

在推断时， f_θ 用于根据语义覆盖度对检索到的候选项重新排序。

B. 使用覆盖验证确保完整性

尽管重新排序提高了候选质量，但检索阶段中的无关信息仍可能导致信息需求仅得到部分覆盖。为了解决这一问题，我们引入一种覆盖验证函数，该函数采用缺口检测与细化算法。

连通表组构建。受基于斯坦纳树的模式图[50]启发，我们通过为每个子问题选择一个表来构建连通表组，使相应的簇在 $\mathcal{G}$中构成一个连通分量。每个组表示一个能够共同覆盖完整问题的候选表集。我们将组中涉及的所有表串接起来，并对串接后的内容和问题应用f_θ ，以此为每个组评分。

缺口检测与细化。 如果排名最高的组的得分低于预定义阈值，我们便认为覆盖不完整。接下来，我们使用大语言模型按照 [51] 生成一个剩余子问题。然后，我们根据这个新子问题检索并重新排序候选表，并选择一个与当前组相连、联合覆盖得分最高的替代表。

最后，我们根据包含每个表的最佳表现组为其评分，并选择 Top-k 个表用于答案生成。这种设计能够实现更高精确率的检索，既高效又可扩展到大规模表格。

备注。尽管表格对齐问题分解器提供了初步的列级对齐，但这种映射不足以支持稳健的表格检索。我们的实证分析表明，由于存在语义歧义，映射结果有 60% 不准确；另有 40% 的映射不完整，因为没有任何单列与信息需求高度匹配，导致相关表被遗漏。因此，必须使用专门的检索器来识别完整且相关的表集合。

V. 子问题引导推理器

检索到相关表后，一种常见的答案生成 策略是使用序列到序列模型 [3]直接在表上进行推理。然而，这种策略在数值问题上的表现往往不佳，因为答案必须通过明确计算（例如标准聚合和算术运算）得出，而不能仅靠简单查找获得。基于程序的推理[15] 提供了一种结构化的替代方案——首先生成可执行程序（例如 SQL 或 Python），然后在多张表上执行该程序以得出最终答案。然而，现有的大多数程序生成器都是为标准的单数据库文本到SQL场景设计的，其中具有明确的主键-外键约束。将其用于我们的场景，即一组彼此孤立且没有明确表间关系的表时，往往会变得不可靠：它们可能会选择无关的表，无法推断哪些表可以连接，或生成错误的推理步骤 [16]。

为应对这些挑战，我们提出了**子问题引导推理器**。我们的方法并非一次性生成完整程序，而是通过一系列分解后的子问题逐步构建程序。每个子问题都可以使用单个表或可合并组可靠地回答，而子问题间依赖关系则决定了应将哪些中间结果连接起来，以得到完整的推理程序。随后执行生成的程序，得出最终答案。这种设计带来了更准确的推理流程，能够支持针对彼此独立的检索到的表进行多种类型的推理程序。

A. 程序生成

给定原始问题、一组分解后的子问题以及检索到的表，我们的推理器分两个阶段生成程序：

思维链引导的多步程序生成。基于多跳推理领域的最新进展 [52], ，我们结合思维链（CoT）提示，根据子问题序列逐步生成程序。该过程首先生成一个初始子程序，该子程序使用与初始子问题相关的表可合并组。随后，对于每个后续子问题，通过将当前中间程序与为该子问题生成的

新子程序连接起来，逐步改进程序。处理完所有子问题后，即得到最终程序。

这一逐步过程带来两项益处：(1) 它基于表依赖关系显式编码推理，以解决高复杂度任务；(2) 它通过基于表集合中更小、连贯的子集构建每个推理步骤，使程序构建更加鲁棒。为处理连接可合并表时的列级不一致，我们还加入了模糊连接算子 [53] ，用于对齐的文本连接键。

执行引导的改进。尽管使用了精心构建的提示，生成的程序仍可能包含错误（例如语法错误）。为提高鲁棒性，我们在方案中引入了执行引导的改进 步骤。我们首先使用检索到的表执行该程序，并检查输出是否存在任何失败。若检测到错误，我们会重新提示大语言模型，并附上错误信息以帮助优化程序。该过程会重复进行，直到生成有效程序或达到最大重试次数。

VI. 评估数据准备

在本节中，我们首先考察现有基准数据集在我们问题设定下的局限性。然后，我们介绍为准备评估数据集而设计的解决方案。

A. 动机

据我们所知，目前还没有基准数据集能够直接支持在大规模表集合上评估数值型多表问答。首先，常用于多表问答评估的数据集 [1], [14]直接来源于文本到SQL基准测试，例如 Spider [17],和 Bird [18]。这些原始及衍生基准数据集通常由一个或多个数据库中的表构成，规模有限、元数据完整，且仅关注显式的主键-外键连接。因此，它们无法反映大规模表集合中的可扩展性、不完整元数据以及复杂表关系。其次，广受欢迎的表格问答基准数据集，例如 NQ-Tables [12] 和 Open-WikiTables [54],，是为开放域单表问答而设计的，即每个问题都可以用单个表来回答。因此，它们缺少用于评估的多表问题-答案对。

B. 准备流程

为弥补这些不足，我们的数据准备分为两个步骤。首先，我们收集明确以多个相关表为依据的真实世界问答对。其次，我们将这些依据表扩展为大规模表格库，以体现大规模表集合的可扩展性、不完整元数据和复杂表关系特征。为此，我们重新利用现有文本到SQL基准测试 Spider [17] 和Bird [18]中的问答对及其依据表。由此得到的数据集称为 SpiderWild 和 BirdWild。接下来，我们将介绍如何通过重新利用依据表并引入外部表来构建大规模表集合，以及如何进行问题标注。

1）重新利用依据表：对于每个文本到SQL基准测试，我们首先从原始数据库中收集所有表，将其作为依据表存入集中式存储库。为体现其独特特征，我们随后应用一个分为三个阶段的表转换流程：

阶段 1：表分解。为克服原始数据集规模有限、表间关系多样性不足的问题，我们借鉴 [19] 的思路，采用表分解方法。与其随机分解不同，我们的分解旨在生成语义上更有意义的表。具体而言，我们将大型表（即列数超过 5 且行数超过 50 的表）通过列方向和行方向的拆分策略，分解为多个互不重叠的子表。这一设计源于数据湖中常见的组织模式：表通常由语义相关的列（例如年份、月份、日期）构成，或按值范围或类别进行分区 [55], [56]。

对于按列拆分策略，我们首先识别键列（即所有值均互不相同的列），并使用大语言模型将其余非键列分组为语义相关的列子集。为此，我们指示大语言模型根据列标题按主题或题材将列聚为簇。随后，将每个列组与随机选取的一个键列组合，形成新的子表。为确保分解后的子表能够通过连接重建原始表（即原始表中同一行的值仍保持正确对齐），我们检查是否存在包含全部键列的子表。若不存在这样的子表，则创建一个包含全部键列的子表。由于原始数据库表仅提供主键-外键连接，我们旨在引入语义连接的额外可连接性。因此，我们选择由同一不参与主键-外键连接的键列派生的子表来建立可连接性。

按行拆分策略在按列拆分步骤之后应用，它根据随机选取的一个非键列的分布对每个子表进行划分。若选取的是数值列，我们根据值范围将行划分为随机数量的桶（介于5到20之间）（例如，按销售额的值范围拆分销售表）。若选取的是类别列，我们随机将表划分为2到20张表，每张表包含某一特定类别组的行（例如，按地区拆分销售表可能会得到“北美销售”“欧洲销售”等独立的表）。这种按行拆分会产生共享相同标题但涵盖不同表内容的可合并表，这反映了现实世界中可合并表的常见情形，如维基百科表的数据版本管理 [57]所示。分解之后，我们人工检查子表，以验证它们保留了原始语义（例如，原始表可通过连接或对子表取并集来重建）。

阶段 2：元数据不完整性模拟。为模拟不完整元数据，我们随机选取 20% 的分解表，并根据 [5], [6] 中报告的不完整元数据统计数据，使用占位符 MASK 隐去其中 50% 的列标题和表标题。例如，

一个包含 emp_id、department、salary 和 role 列的员工表，可能会被转换为一个标题为 MASK、包含 MASK、department、MASK 和 role 列的表。

阶段 3：可连接性模拟。在各种类型的可连接性中，模糊连接（值的拼写或格式略有差异）在真实世界数据湖中尤为常见[58]–[60]。为模拟这一情况，我们在分解表的文本键列中引入值级变化。具体而言，按照 [60]，，我们在 20% 的单元格值中加入拼写错误和字符删除等扰动。这种转换创造了逼真的连接场景，需要进行近似字符串匹配（例如，“纽约”与“Nw York”）。

2）纳入外部表：虽然我们的表分解策略将表的数量从数百个增加到数千个，但仍不足以反映实践中通常遇到的大规模场景，例如线上可获取的大量网页表格集合。为进一步提升可扩展性，我们从广泛使用的表语料库 WebTables 和 OpenData 中纳入额外的表[19]，[61]，并考虑两个维度：（1）与问题在语义上相关的表——作为困难负例；（2）与现有表属于相似领域的表，其中领域相关性通过表标题的相关性进行估计——以模拟真实的干扰表。具体而言，对于每个收集的问题或分解表，我们使用 BM25 [62] ，针对问题和表名称，从这些语料库中检索排名前N 的候选表。这种扩充显著扩大了表库。例如，在 N=100 的数据集上，SpiderWild 中的表数量从 2,210 个增至 73,688 个，BirdWild 中则从 5,136 个增至 109,949 个。

3) 问题标注： 最后，我们从原始问题中筛选出一组需要跨多个表进行推理的数值问题。对于每个选定的问题，我们都会构建标注，其中包括真实答案以及回答该问题所需的相关表集合。具体而言，我们首先在提供的数据库上执行原始 SQL 查询，以获得真实答案。随后，我们在分解表上通过连接/并集运算编写并执行改写后的 SQL 查询，以验证答案是否一致。接下来，为了确定相关表，我们分析原始 SQL 查询如何从多个表中提取信息。为此，我们手动改写查询，以提取每个相关表中主键及其他相关列（例如 WHERE 或 GROUP BY 子句中提及的列）的值。然后，我们找出包含这些值的具体表行和列。接着，通过定位包含相应行和列的表，将这些记录映射回表库中的分解表。能够共同涵盖所有这些记录的分解表子集即视为相关表。我们还会抽查 10% 的问题，以确认外部表不会引入其他可行的解法。

VII. 实验评估

评估目标。我们旨在评估

属性	SpiderWild	BirdWild
# 表	73,688	109,949
% 可连接性	9%	13%
平均列数	4.4	5.0
平均行数	1,384	8,629
数值问题数量	274	461
% 简单/中等/困难	52%/43%/5%	10%/84%/6%
% 相关表（2 / >=3）	67% / 33%	64% / 36%
% 不完整元数据	42%	48%
% 要求可并集性	13%	56%

表 I：数据集统计。

我们提出的方法在大型表集合上的真实多表问答场景中的有效性、效率、可扩展性和鲁棒性，并与文本到SQL和开放域单表问答方法进行次要比较。具体而言，我们回答以下五个问题：

- **问题一：我们的方法在检索多个相关表并回答数值问题方面表现如何？** 不同于以往的开放域多表问答研究，后者难以大规模检索相关表并生成可靠答案（§II-B），我们的框架旨在解决这些局限。我们针对表格检索和答案生成，评估其相较于强基线方法的有效性和效率。（§ VII-A）

- **问题二：我们的关键设计选择和参数设置对整体性能有何贡献？** 我们开展消融研究，以评估核心组件（即预处理、分解器、检索器和推理器）的影响，并分析表可连接性和可并集性阈值等关键参数的敏感性。（§ VII-B）

- **问题三：随着表语料库规模扩大，我们的方法表现如何？** 我们评估随着表语料库中的表数量增加，DMRAL 是否仍能保持较高的有效性和较低的延迟。（§ VII-C）

- **问题四：我们的方法在不同挑战性场景下的鲁棒性如何？** 具体而言，我们评估其相对于所涉表数量变化、表的可并集性以及元数据完整性程度的鲁棒性。（§ VII-D）

- **问题五：我们的方法与文本到SQL及开放域单表问答方法相比如何？** 我们针对文本到SQL（§ VII-E）和开放域单表问答方法（§ VII-F）进行评估。

A. 表格检索与答案生成评估

1) 实验设置： 我们在预处理后的 SpiderWild 和 BirdWild 数据集上评估 DMRAL。有三个主要因素会直接影响多表问答：（1）表连接的数量；（2）是否涉及并集操作；（3）相关表是否包含不完整元数据。为更细致地评估这些因素，我们根据推导答案所需的表连接和并集数量，以及是否涉及不完整元数据，将两个数据集中的问题划分为三个复杂度等级——简单、中等和困难。具体而言，简单 问题涉及两次表连接，不需要并集操作，且依赖完整元数据。相比之下，困难 问题

Top-k方法		SpiderWild									BirdWild								
		Easy			中等			Hard			Easy			中等			Hard		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Top-3	JAR并集	46.5	69.7	55.8	45.2	60.6	51.3	44.8	59.8	50.7	39.7	55.0	45.8	39.2	53.7	45.7	45.0	53.7	44.8
	MMQA并集	47.2	70.8	65.6	37.9	52.4	43.7	37.9	52.1	43.5	41.0	56.6	49.1	36.7	49.7	41.8	35.9	48.8	41.0
	OpenSearch-Wild	43.9	65.8	52.7	42.4	50.1	45.4	36.7	45.0	40.0	49.5	67.4	56.5	40.4	60.6	48.5	36.8	43.0	39.0
	DMRAL	**54.9**	**82.4**	**65.9**	**52.9**	**71.3**	**60.2**	**52.3**	**70.3**	**59.4**	**52.2**	**71.0**	**59.6**	**52.7**	**70.7**	**59.8**	**51.5**	**69.5**	**58.6**
Top-5	JARUnion	31.3	78.2	44.7	32.3	71.5	44.1	31.9	70.4	43.5		28.6	65.6	39.5		28.4	64.4	39.1	28.1	64.1	38.8
	MMQA 并集	31.1	77.8	44.5	26.4	60.3	36.5	26.2	59.7	36.2	26.7	60.9	36.9	26.9	60.7	37.0	26.3	59.6	36.2		
	OpenSearch-Wild	30.6	76.4	43.7	28.3	65.2	39.5	24.0	50.8	32.6	32.1	72.8	44.2	24.8	62.1	35.5	29.0	55.0	37.3		
	DMRAL	**34.2**	**85.6**	**48.9**	**33.1**	**74.4**	**45.4**	**32.7**	**73.5**	**45.0**	**33.3**	**74.8**	**45.4**	**33.3**	**74.7**	**45.7**	**32.8**	**73.7**	**45.0**		

表 II：SpiderWild 和 BirdWild 数据集上的表格检索效果对比。

Top-k方法		SpiderWild			BirdWild		
		简单	中等	困难	简单	中等	困难
Top-3	JARUnion	40.8	28.9	10.0	28.8	24.9	1.2
	MMQAUnion	40.8	25.6	14.0	26.7	24.9	3.8
	OpenSearch-Wild	47.8	42.1	20.0	41.6	41.8	20.0
	DMRAL	**57.0**	**43.8**	**20.0**	**45.7**	**43.3**	**26.9**
Top-5	JARUnion	46.5	38.0	30.0	41.7	36.0	3.8
	MMQAUnion	46.5	33.9	28.0	35.1	31.9	11.5
	OpenSearch-Wild	47.0	40.5	20.0	40.7	39.8	20.0
	DMRAL	**57.7**	**41.3**	**35.0**	**47.5**	**44.6**	**19.2**

表 III：检索到的表上的答案准确率比较。

需要两次以上连接，至少涉及一次并集操作，且相关表包含不完整元数据。不符合简单或困难类别标准的问题归为中等。表 I 总结了数据集统计信息。

评估指标。 遵循先前工作[1], [14],，我们使用精确率@k（P@k）、召回率@k（R@k）和 F1@k 来评估多表检索的有效性。为衡量数值问题的答案准确率，我们采用 [63],中的算术精确匹配（EM@k），该指标衡量使用 Top-k 检索到的表执行生成的程序所得到的、与真实值匹配的答案所占百分比。我们遵循先前研究 [1]，将 k 取为 3 和 5。

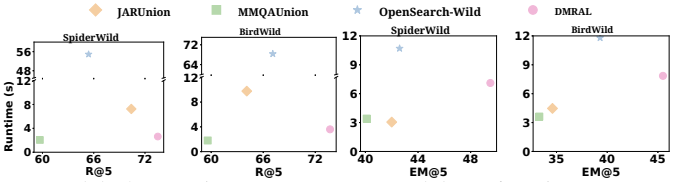
实现与硬件。我们使用 BGE-M3 [64]进行嵌入，因其在检索任务中的有效性 [65]。我们将可并集性和可连接性阈值分别设为 0.9 和 0.5。我们采用 GPT-4.1 mini 作为主要大语言模型。本框架所使用的提示见附录 B。所有实验均在一台运行 Red Hat 7.9 的服务器上进行，该服务器配备 Intel(R) Xeon(R) E5-2690 CPU、512GB 内存和一块 16GB NVIDIA Tesla P100 GPU。我们的源代码和基准数据集可在 [66] 获取。

竞争方法。 我们首先将所提出的方法与 JARUnion 和 MMQAUnion 进行比较；它们分别改编自当前最先进的开放域多表问答系统 JAR [14]和 MMQA [1], 。这两种方法最初都是为具有完整元数据的数据库设计的，并未考虑表可合并性。为确保比较公平，我们在经过处理的元数据完整的表上应用其原始表检索策略，并通过纳入可合并性来扩充检索到的表（详情见附录 C）。

2）多表检索的有效性： 表 II 展示了不同问题复杂度下的表检索效果。我们重点指出两个主要观察结果：（1）在两个数据集和所有复杂度级别上，DMRAL 的表现始终优于所有基线方法。这一提升主要得益于我们的分解驱动多表检索策略：分解器生成质量更高的子问题，随后检索器通过明确最大化问题覆盖率，选出一组连贯的表。相比之下，JARUnion 和 MMQAUnion 缺乏有效的多表检索机制。（2）随着问题复杂度上升，几乎所有方法的性能都有明显下降。这些趋势表明，当问题需要更多可连接和可合并表时，检索相关表会变得更加困难，尤其是在元数据不完整的情况下。

3）答案生成的有效性：表 III 展示了使用 Top-k 检索结果时的答案准确率。我们按问题复杂度分层，报告了 Top-3 和 Top-5 检索设置下的精确匹配分数。我们有以下几项重要发现：（1）在两个数据集的所有复杂度级别上，DMRAL 的表现始终优于所有基线方法，这得益于更准确的表格检索和程序生成。（2）随着问题复杂度增加，两个数据集上的答案准确率也显著下降。这表明，更复杂的问题——涉及更多连接、并集和不完整元数据的问题——不仅在识别正确的表方面更具挑战，在通过推理生成正确答案方面也更具挑战。（3）有趣的是，不同复杂度级别之间答案准确率的降幅（例如，BirdWild 上从简单到困难时，EM@3 下降了 70%）远大于多表检索的降幅（例如，R@3 下降了 2%）。这表明，即使表格检索准确，随着连接、并集和不完整元数据的复杂度增加，对多个表进行推理也会变得更具挑战性。附录 D 对表格检索和答案生成进行了系统的错误分析。

4）多表检索与答案生成的效率：为评估多表问答方法的效率，我们测量了各数据集中所有问题的多表检索和答案生成平均耗时。请注意，对于 DMRAL 的多表检索运行时间，我们将问题分解器和检索器模块所耗费的时间都计算在内。图3对效率及相应的有效性进行了比较分析，此外



（a）检索效率　　（b）答案效率

图3：表格检索和答案生成的效率分解，以每个问题的平均运行时间进行衡量。

Top-k	方法	SpiderWild P	SpiderWild R	SpiderWild F1	SpiderWild EM	BirdWild P	BirdWild R	BirdWild F1	BirdWild EM
Top-3	不填充	42.5	55.8	46.0	42.6	40.9	57.0	47.6	28.4
	直接大语言模型	50.3	67.0	57.4	44.5	48.6	66.2	56.0	40.3
	DMRAL	52.3	70.3	59.4	49.0	51.5	69.5	58.6	44.3
	GT	59.1	78.0	67.3	51.9	57.2	76.5	65.4	45.2
Top-3	不填充	25.3	59.8	35.7	43.0	26.7	58.5	36.6	29.2
	直接大语言模型	31.4	70.0	43.4	45.0	30.9	70.2	43.0	41.4
	DMRAL	32.7	73.5	45.0	49.5	32.8	73.7	45.0	45.5
	GT	37.0	81.6	50.9	52.5	36.4	81.1	50.2	46.4

表 IV：元数据质量对表格检索和答案生成有效性的影响。

我们观察到以下结果：（1）对于表格检索，DMRAL 在有效性与延迟之间实现了良好平衡。与效率最高的基线 MMQAUnion 相比，它在两个数据集上的 R@5 平均高出 15%，而运行时间仅增加 1.6×。（2）对于答案生成，DMRAL 显著提升了准确率：与次优方法 JARUnion 相比，它在 SpiderWild 和 BirdWild 上的 EM@5 分数分别高出 18% 和 31%，代价是每个问题平均增加 3.7 秒。这部分额外耗时源于 DMRAL 推理组件中采用的两种面向推理的机制，它们旨在增强程序的鲁棒性：（i）采用逐步思维链提示策略，逐步生成推理程序；（ii）采用优化机制，在执行前验证并改进生成的程序。

B. 消融研究和参数研究

1) 消融研究：我们进行了全面的消融研究，以评估关键设计——即预处理（§II-C）、分解器（§III）、检索器（§IV）和推理器（§V）组件——对有效性的影响。

预处理。由于真实世界的表语料库可能包含不完整元数据，这会在图构建过程中扭曲可并集性计算，我们还引入了元数据推断模块。该模块利用大语言模型，通过结合表内上下文来推断缺失的列标题，其灵感来自 [67]（详情见附录 A）。

接下来，我们将评估该模块的推断质量，并与朴素基线直接大语言模型进行比较。该基线直接提示大语言模型补全缺失的元数据，而不使用表内上下文。按照先前的研究 [36]，我们使用*BERT-F1*分数评估质量：对于每个缺失的列标题，我们根据上下文嵌入相似度计算预测标题与真实标题之间的 F1 分数，并报告所有缺失标题的平均值。我们的方法始终优于直接大语言模型：在 SpiderWild 上，BERT-F1 从 0.602 提升至 0.697；在 BirdWild 上，则从 0.602 提升至 0.652。这些提升表明，利用表内上下文能够显著提高元数据推断的准确性。

方法	SpiderWild IRR（↑）	SpiderWild SR（↓）	SpiderWild SAR（↑）	BirdWild IRR（↑）	BirdWild SR（↓）	BirdWild SAR（↑）
直接大语言模型	93%	0.589	57%	94%	0.600	59%
DMRAL	96%	0.521	71%	96%	0.522	70%

表 V：分解质量比较。↑：越高越好，↓：越低越好。

我们进一步评估元数据质量如何影响下游检索和答案准确率，比较了四种配置：（1）不填充，即保持不完整元数据不变；（2）直接大语言模型；（3）DMRAL，即我们提出的元数据推断模块；（4）真实值，即使用相应的真实元数据替换所有不完整元数据。结果见表 IV，涵盖 Top-3 和 Top-5 两种检索设置。从结果中我们观察到，在 SpiderWild 和 BirdWild 数据集上，DMRAL 在所有评估指标上均持续优于不填充和直接大语言模型基线方法。这些结果凸显了补全缺失元数据的重要性，以及利用表内上下文进行元数据恢复的有效性。此外，与不填充相比，DMRAL 与真实值设置之间的性能差距相对较小。这表明我们的推断策略能够紧密逼近黄金标准元数据，并对多表问答中的表格检索和答案生成均有显著贡献。

表格对齐问题分解器。我们首先将我们的分解策略与直接大语言模型 生成方法进行比较。该方法使用大语言模型生成子问题，但不提供任何结构对齐信息。为评估分解质量，我们采用三个指标，分别对应 §III 中定义的标准：（i）信息保留率（*IRR*），用于衡量完整性：分解成功保留回答问题所需的全部信息需求的问题所占比例；（ii）子问题冗余度（*SR*），用于衡量冗余度：使用预训练 Sentence-BERT 模型生成的嵌入计算出的两两语义相似性的平均值；（iii）子问题-表格对齐率（*SAR*），用于衡量表特异性：判断子问题的数量是否与回答问题所需的相关表数量相符。

从表 V 可以看出，使用两个数据集时，我们的表格对齐问题分解器在全部三个指标上均持续取得提升。这些结果证明了我们利用表语料库结构的分解策略是有效的。附录 E 展示了进一步说明这些改进的案例研究。

接下来，我们在表 VI 中进一步深入研究分解对检索效果和最终答案准确率的影响。

Top-k	方法	SpiderWild				BirdWild			
		P	R	F1	精确匹配	P	R	F1	精确匹配
Top-3	直接大语言模型	50.9	68.3	57.8	45.4	49.5	67.2	56.5	38.9
	DMRAL	**52.3**	**70.3**	**59.4**	**49.0**	**51.5**	**69.5**	**58.6**	**44.3**
Top-5	直接大语言模型	31.5	70.7	43.2	49.1	32.0	72.1	43.9	39.6
	DMRAL	**32.7**	**73.5**	**45.0**	**49.5**	**32.8**	**73.7**	**45.0**	**45.5**

表 VI：使用我们的分解器与 directLLM分解器进行有效性比较。

方法	SpiderWild		BirdWild	
	R@3	R@5	R@3	R@5
DMRAL	**70.3**	**73.5**	**69.5**	**73.7**
朴素评分	62.6 (−11)	66.7 (−9)	66.4 (−4)	71.3 (−3)
无验证	56.3 (−20)	60.7 (−17)	55.9 (−20)	64.7 (−12)

表 VII：覆盖感知检索器的消融研究[1]。

观察表 VI 可知，我们的分解方法在 Top-3 和 Top-5 检索设置下始终表现更佳。这些结果表明，DMRAL 通过提升分解后的子问题质量，同时改善了检索效果和最终答案准确率。

覆盖感知检索器。现在，我们通过两项消融实验评估覆盖度评分函数和覆盖验证子模块的检索效果：(1)朴素评分使用一个简单的基线方法替代我们训练的覆盖度评分函数，该方法将子问题与候选表之间的各项嵌入相似度相加，这些相似度由前文所述的粗粒度检索方法计算得出。(2) 无验证 禁用用于检索互补表格的剩余子问题生成模块。表 VII 显示：(1) 我们训练的覆盖度评分函数能够识别出更全面覆盖问题意图的表子集，从而带来更有效的检索效果。(2) 我们的覆盖验证子模块通过弥补无关表引入的潜在覆盖缺口，提升了检索效果——这些表看似与问题相关，却无法提供完整答案。

子问题引导推理器。我们通过与另外两种变体进行比较，评估推理模块的效果：(1) 无思维链 基线提示大语言模型仅根据原始问题和检索到的最相似表，一次性生成程序；(2) 无优化执行生成的第一个程序，不再进行进一步优化。答案准确率如表 VIII 所示，结果表明：(1) 我们的思维链生成显著提升了答案质量，精确匹配分数最高提高了 19%。这表明，通过子问题对推理过程建模大有裨益；这种方法使大语言模型能够正确推断表关系，从而生成准确的程序并给出可靠的答案。(2) 我们的执行引导改进通过确保生成的程序正确，进一步提升了答案质量。

2) 参数研究：我们开展参数研究，以探究不同的可连接性阈值对表关系建模（§ II-C）的影响

[1]括号中的数值表示相对于我们完整方法的相对性能下降。

方法	SpiderWild		BirdWild	
	EM@3	EM@5	EM@3	EM@5
DMRAL	**49.0**	**49.5**	**44.3**	**45.5**
无思维链	44.8 (−10)	45.2 (−10)	34.1 (−23)	37.0 (−19)
无优化	48.3 (−3)	49.7 (−1)	37.8 (−10)	42.0 (−8)

表 VIII：我们的推理器的消融研究。

以及可并集性阈值对表关系建模（§ II-C）的影响。我们将可连接性的阈值范围设为 [0.1、0.3、0.5、0.7、0.9] ，将可并集性的阈值范围设为 [0.5、0.6、0.7、0.8、0.9] 。这些下限是根据其在所有表对中观察到的显著剪枝效果选定的，能够有效过滤掉大量低质量或虚假的表间关系。图 5 展示了不同阈值设置下的检索与回答有效性。可以看到：(1) 对于可连接性，随着阈值从 0.1 增至 0.5，两项性能均有所提升，之后则有所下降。这是因为极低的阈值（例如 0.1）会纳入许多无关表之间的噪声或虚假连接，从而降低检索质量，并为推理引入无关候选项。而极高的阈值（例如 0.9）则过于严格，会排除中等程度可连接的表，而这些表对于推导出正确答案仍然必不可少。(2) 对于可并集性，随着阈值提高，两项性能均持续提升。这是因为更严格的阈值确保只有高度可合并的表（即语义相似性很强的表）才会被归为一组。这减少了语义不匹配的表被纳入并集组的情况，从而提升检索精确率以及推理的依据质量。

C. 可扩展性研究

1) 实验设置：为了评估 DM-RAL 在不同表语料库规模下的可扩展性，我们将每个问题/表格检索的前 N 个外部表的数量（§ VI-B2）设置为 $N \in \{100, 150, 200, 250, 300\}$。这些设置分别生成约含 109K、149K、183K、214K 和 243K 个表的表语料库。由于表语料库的规模在很大程度上影响表格检索的效果和效率，我们主要关注以 R@5 衡量的检索效果，以及以每个问题的检索耗时衡量的检索效率。

2) 主要结果：图 5 展示了结果。随着表语料库规模扩大，我们发现 DMRAL 的检索效果依然稳健——尽管表的数量增加了一倍多，R@5 也仅下降了 5%。在效率方面，查询时间随规模逐步增长，表明即使面对更大的表语料库，该方法仍能保持较高的计算效率。在离线图构建效率方面，对于两个数据集，使用规模最大的 243K 张表计算可连接性（使用 5 个并行工作进程）约需 4.2 天，而计算可并集性约需一天，因为其处理的是表对，数量少于计算可连接性所需的列对。展望未来，我们计划借助分布式工程技术（例如分片和并行相似连接）降低预处理成本 [68]。我们还计划支持可扩展、低成本的动态更新

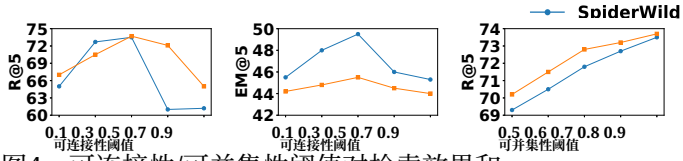


图4：可连接性/可并集性阈值对检索效果和答案准确率的影响。

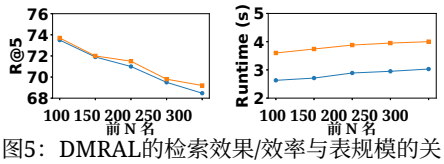
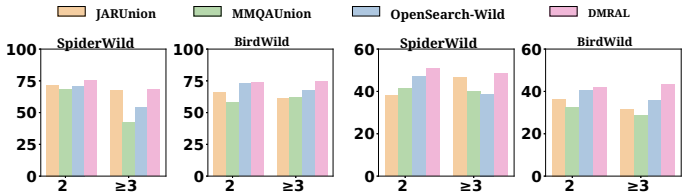
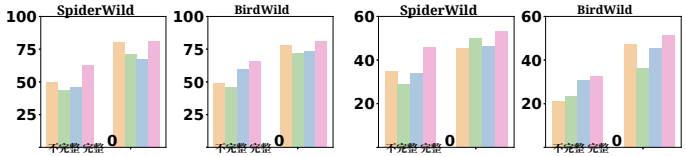


图5：DMRAL的检索效果/效率与表规模的关系。



（a）检索：R@5　　（b）答案：EM@5

图 6：相关表数量（2 与 ≥3）对有效性的影响。



(a) 检索：R@5　　(b) 答案：EM@5

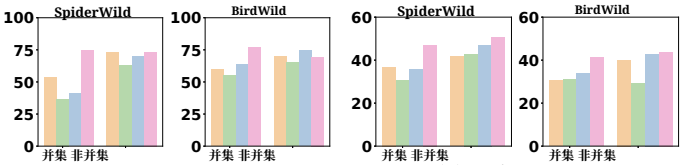
图 7：对涉及元数据不完整与完整的表的问题进行评估。

以便在无需完全重新计算的情况下，增量式地纳入新表及其关系。

D. 鲁棒性跨挑战场景

需要不同数量相关表的问题上的鲁棒性。为评估涉及不同数量表的问题上的鲁棒性，我们根据检索到的相关表数量（2 与 ≥3）对问题进行分组，并评估检索和答案效果。如图 6 所示，DMRAL 在两种分组中始终优于所有其他基线方法，展现了我们的方法在复杂度更高的问题上的鲁棒性。这是因为：（1）我们的检索器明确优化问题覆盖率，确保即使所需表集合不断扩大，也能选出必要的表；（2）我们的推理器生成由分解后的子问题引导的程序，从而能够在难度增加时准确推断表关系。

涉及不完整元数据的问题的鲁棒性。我们还通过比较含有不完整元数据的相关表的问题与元数据完整情况下的问题表现，考察了我们方法的鲁棒性。结果如图 7 所示。观察可知：（1）引入需要使用元数据不完整的表的问题后，所有方法的性能均有所下降。这凸显了不完整元数据带来的挑战。（2）在两个数据集上，DMRAL 在检索和答案效果方面始终是最有效的方法，与基线方法相比，性能下降幅度明显更小（例如，DMRAL 的 EM@5 平均下降 26%，而基线方法下降 62%）



(a) 检索：R@5　　（b）答案：EM@5

图 8：评估需要整合可合并表（并集）的问题与不需要此类表的问题。

）。这些结果表明，存在不完整元数据时，DMRAL 更具鲁棒性。

需要可合并表的问题上的鲁棒性。现在，我们比较了使用需要可合并表才能回答的问题与不需要此类表的问题时，本方法的性能。结果如图 8 所示。再次证明，DMRAL 始终是效果最佳的检索模型，并且在两种问题类型和两种数据集上都能给出更高质量的答案。现有基线方法在回答需要执行一个或多个并集操作的问题时，性能会显著下降；而 DMRAL 在这两种情形下始终表现出色。这一优势源于我们的设计决策：通过图对表可合并性进行显式建模，并在检索与推理阶段将可合并表归入簇。相比之下，扩展基线方法假设每张表彼此独立，未能将可合并性纳入检索模型，因此难以找到多个可连接表并利用它们进行推理。

E. 与文本转*SQL*方法的比较

文本转SQL方法是为具有完整元数据和显式表关系的数据库而设计的。这一设定提供了我们场景中所不具备的优势。为进行公平比较，我们开展两项互补评估：（1）通过数据库物化，在相同集合SpiderWild和BirdWild上运行文本到SQL的适配评估（§VII-E1）；（2）在其原始设定下进行的二次评估，其中文本转SQL方法获得完整的数据库模式，而DMRAL仅获得针对每个问题从相应关系数据库中收集的表集合（§VII-E2）。

1）SpiderWild 与 BirdWild 上的适配评估：为了在大规模表集合 SpiderWild 和 BirdWild 上进行公平比较，我们将 DMRAL 与一个适配后的文本到SQL基线 OpenSearch-Wild 进行比较，该基线基于高性能的 OpenSearch-SQL [28]，，如 [69] 所报告。由于 OpenSearch-SQL 在单个数据库上运行，且不包含表检索阶段，我们通过添加以下内容将其适配到我们的设定中：(i) 在其预处理过程中，基于我们的推断元数据和表格关系进行数据库物化

图，以及 (ii) 在其提取过程中基于其所获得的列进行 top-k 表选择（详情见附录 C）。

主要结果。 我们总结改编后的基线方法 OpenSearch-Wild 和 DMRAL 在有效性、效率和鲁棒性方面的表现如下。(1) 如表 II 所示，DMRAL 的表现始终优于 OpenSearch-Wild。这是因为 OpenSearch-Wild 依赖问题-列相关性，容易受到列映射错误的影响；而 DMRAL 通过明确最大化问题覆盖率来构建连贯的表集合，从而减少级联检索错误。(2) 如表 III 所示，OpenSearch-Wild 的表现仍落后于 DMRAL。这主要是因为其较弱的表格识别能力常常会漏掉 SQL 生成所需的表。(3) 如图3所示，OpenSearch-Wild 在检索和答案生成两个阶段的延迟都是最高的。在检索阶段，它增加了一个线上列筛选环节，选出与问题的语义相似性超过阈值的列，这需要进行嵌入计算和相似度评分。在答案生成阶段，它采用多轮分步提示，在生成最终 SQL 之前推导 SQL 元素（例如 SELECT子句和值）。(4) 在涉及表数量、并集能力和元数据完整性各不相同的情况下（图6–8），DMRAL 的表现更稳定，这得益于其鲁棒的多表检索与推理框架。

2）使用原始设置进行次级评估：我们在三个广泛使用的文本到SQL基准测试上进行评估——Spider [17], BIRD-dev [18],和 ScienceBenchmark [70]。对于每个基准测试，我们聚焦于经过筛选的数值问题子集，这些问题需要跨多个表进行推理。Spider 共包含 274 个数值问题，涉及 79 个数据库和 437 张表；BIRD-dev 包含 45 个数值问题，涉及 7 个数据库和 52 张表；ScienceBenchmark 包含 36 个数值问题，涉及 3 个数据库和 50 张表。我们将方法与两种资源效率高、性能出色的文本到SQL系统 CHESS 和 OpenSearch-SQL（§II-B）进行比较，相关结果见 [69]。我们报告两项指标：表格选择准确率，即预测 SQL 恰好引用真实相关表的问题所占百分比；以及答案准确率，该指标对应于文本到SQL中的执行准确率 [17]。

主要结果。 如表 IX 所示，DMRAL 在表格识别方面表现强劲：它在 Spider 和 ScienceBenchmark 上表现最佳，并在 BIRD-dev 上与表现最佳的 OpenSearch-SQL 相比仍具有竞争力。这表明，尽管无法获取主键-外键约束，DMRAL 仍能有效识别正确的表子集。就答案准确率而言，DMRAL 在 Spider 和 ScienceBenchmark 上表现最佳。我们将这些提升归因于其更强的表格识别能力，从而为推理器提供更完整的相关表。在 BIRD-dev 上，我们观察到与 OpenSearch-SQL 相比存在差距。我们的错误分析表明，大多数失败（57%）源于值 grounding，即生成的 SQL 中的值与列值不匹配。这在预期之中，因为我们的重点是面向大规模表的稳健多表检索与回答，而非优化 SQL生成。我们的推理器未包含值映射模块，原因在于大量的值预处理和存储开销，而 OpenSearch-SQL 则纳入了用于值 grounding 的值级提取。在未来的工作中，我们将通过轻量级值映射增强推理器，以提高 SQL 正确性。

方法	Spider		Bird-dev		ScienceBenchmark	
	表选择	答案	表选择	答案	表选择	Ans.
DTR	0.586	0.490	0.422	0.400	0.200	0.129
TableRAG	0.667	0.597	0.600	0.267	0.294	0.118
CHESS	0.769	0.801	0.733	0.733	0.324	0.306
OpenSearch-SQL	0.777	0.822	**0.880**	**0.867**	0.482	0.400
DMRAL	**0.864**	**0.826**	0.867	0.778	**0.500**	**0.441**

表 IX：在原始设置下与多表问答方法进行比较，采用表选择和答案准确率。

F. 与开放域单表问答方法的比较

在这项比较中，我们使用与 DMRAL（§VII-E2）相同的三个数据集。我们将 DTR [12], （一种稠密表格检索器）与基于大语言模型的程序化推理器 [14], 配对的方法，以及 TableRAG [38],（一种迭代式检索增强生成方法，该方法以检索到的表集合为条件输入大语言模型来生成答案）作为比较对象。我们报告表格选择准确率和答案准确率。表格选择准确率定义为检索到的表集合与金标准相关表格完全匹配的问题所占比例；答案准确率定义为预测答案与金标准答案匹配的问题所占比例。

主要结果。 表 IX 显示，DMRAL 在表格识别和答案准确率方面均显著优于两种单表问答基线方法，凸显了对多表问答中的表间关系进行建模的重要性。

VIII. 结论

本文提出了 DMRAL，这是一种新颖的、由分解驱动的多表检索与问答框架，专为大规模表集合上的数值型多表问答而设计。DMRAL 包含四个模块：预处理模块用于构建图，以捕获表之间的复杂关系；表格对齐问题分解器和覆盖感知检索器通过提升分解质量并最大化覆盖范围，协同改进检索效果；子问题引导推理器则利用引导式大语言模型，根据分解后的子问题生成准确、可执行的程序，从而提升答案质量。在两个构建的数据集上进行的实验表明，DMRAL 显著优于现有最先进的多表问答方法，在表格检索方面平均提升 24%，在答案准确率方面平均提升 55%。

参考文献

[1] J. Wu, L. Yang, D. Li, Y. Ji, M. Okumura 和 Y. Zhang，“Mmqa：使用多表多跳复杂问题评估大语言模型”，载于第十三届国际表示学习大会，2025年，第1页。[2] Z. Qiu, Y. Peng, G. He, B. Yuan 和 C. Wang，“Tqa-bench：通过可扩展上下文和符号扩展评估大语言模型的多表问答能力”，*arXiv* 预印本 *arXiv:2411.19504*，2024年。

[3] V. Pal, A. Yates, E. Kanoulas 和 M. de Rijke，“Multitabqa：为多表问答生成表格答案”，载于《计算语言学协会第61届年会论文集》（第1卷：长篇论文），2023年，第6322–6334页。[4] R. Nararatwong, C.-C. Chen, N. Kertkeidkachorn, H. Takamura 和 R. Ichise，“Dbqr-qa：一个融合数据库查询与推理的问答数据集”，载于《计算语言学协会 ACL 2024 研究成果》，2024年，第15 169–15 182页。[5] Q. Wang 和 R. C. Fernandez，“Solo：通过自监督方法使用自然语言问题进行数据发现”，《ACM 数据管理会议录》，第1卷，第4期，第262:1–262:27页，2023年。[6] A. Khatiwada, G. Fan, R. Shraga, Z. Chen, W. Gatterbauer, R. J. Miller 和 M. Riedewald，“Santos：基于关系的语义表联合搜索”，《ACM 数据管理会议录》，第1卷，第1期，第1–25页，2023年。[7] G. Fan, J. Wang, Y. Li, D. Zhang 和 R. J. Miller，“利用上下文化列表示学习，从数据湖中发现语义感知数据集”，《VLDB 基金会会议录》，第16卷，第7期，第1726–1739页，2023年。[8] Y. Dong, C. Xiao, T. Nozawa, M. Enomoto 和 M. Oyamada，“Deepjoin：利用预训练语言模型发现可连接表”，《VLDB 基金会会议录》，第16卷，第2470页，2023年10月，第2458页。[9] B. Li, J. Zhang, J. Fan, Y. Xu, C. Chen, N. Tang 和 Y. Luo，“Alpha-sql：利用蒙特卡洛树搜索实现零样本文本到SQL”，arXiv 预印本 arXiv:2502.17248，2025年。[10] A. Mohammadjafari, A. S. Maida 和 R. Gottumukkala，“从自然语言到 SQL：基于大语言模型的文本到 SQL 系统综述”，arXiv 预印本 arXiv:2410.01066，2024年。[11] P. B. Chen, F. Wenz, Y. Zhang, D. Yang, J. Choi, N. Tatbul, M. Cafarella, C.. Demiralp 和 M. Stonebraker，“Beaver：文本到 SQL 的企业级基准测试”，arXiv 预印本 arXiv:2409.02038，2024年。[12] J. Herzig, T. Mueller, S. Krichene 和 J. Eisenschlos，“通过稠密检索实现表格开放域问答”，载于《2021年计算语言学协会北美分会年会：人类语言技术论文集》，2021年，第512–519页。[13] F. Pan, M. Canim, M. Glass, A. Gliozzo 和 J. Hendler，“ 通过检索增强生成实现端到端表格问答 ”，arXiv 预印本 arXiv:2203.16714，2022年。[14] P. B. Chen, Y. Zhang 和 D. Roth，“表格检索问题是否已解决？探索连接感知多表检索”，载于《计算语言学协会第62届年会论文集》（第1卷：长篇论文），2024年，第2687–2699页。[15] J.-P. Zhu, P. Cai, K. Xu, L. Li, Y. Sun, S. Zhou, H. Su, L. Tang 和 Q. Liu，“Autotqa：通过多智能体大语言模型实现自主表格问答”，《VLDB 基金会会议录》，第17卷，第12期，第3920–3933页，2024年。[16] J. Shen, C. Wan, R. Qiao, J. Zou, H. Xu, Y. Shao, Y. Zhang, W. Miao 和 G. Pu，“基于上下文学习的文本到 SQL 错误研究”，CoRR，卷 abs/2501.09310，2025年。[17] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang 和 D. R. Radev，“Spider：用于复杂跨领域语义解析和文本到 SQL 任务的大规模人工标注数据集”，载于 EMNLP。计算语言学协会，2018年，第3911–3921页。[18] J. Li, B. Hui, G. Qu, B. Li, J. Yang, B. Li, B. Wang, B. Qin, R. Cao, R. Geng 等，“大语言模型能否胜任数据库接口？用于大规模数据库接地文本到 SQL 的大型基准”，arXiv 预印本 arXiv:2305.03111，2023年。[19] Y. Deng, C. Chai, L. Cao, Q. Yuan, S. Chen, Y. Yu, Z. Sun, J. Wang, J. Li, Z. Cao 等，“Lakebench：用于发现数据湖中可连接和可合并表的基准测试”，《VLDB 基金会会议录》，第17卷，第8期，第1925–1938页，2024年。[20]F. Li 和 H. V. Jagadish，“Nalir：用于查询关系数据库的交互式自然语言接口”，载于国际数据管理会议，SIGMOD 2014，美国犹他州斯诺伯德，2014年6月22日至27日，C. E. Dyreson、F. Li 和 M. T. ¨Ozsu 编。ACM，2014年，第709–712页。[21] H. Fu, C. Liu, B. Wu, F. Li, J. Tan 和 J. Sun，“Catsql：迈向真实世界的自然语言到 SQL 应用”，《VLDB 基金会会议录》，第16卷，第6期，第1534–1547页，2023年。[22] T. Yu, M. Yasunaga, K. Yang, R. Zhang, D. Wang, Z. Li 和 D. Radev，“Syntaxsqlnet：面向复杂及跨领域文本到 SQL 任务的语法树网络”，arXiv 预印本 arXiv:1810.05237，2018年。[23] X. Liu, S. Shen, B. Li, P. Ma, R. Jiang, Y. Zhang, J. Fan, G. Li, N. Tang 和 Y. Luo，“大语言模型时代的文本到 SQL 综述：我们身处何处，又将走向何方？”《IEEE知识与数据工程汇刊》，第 01 期，第 1–20 页，2025年。[24] C. Xiao, M. Dymetman 和 C. Gardent，“用于语义解析的基于序列的结构化预测”，载于计算语言学协会年会（ACL），2016年，第 1341–1350 页。[25] P. Yin, G. Neubig, W. Yih 和 S. Riedel，“TaBERT：面向文本与表格数据联合理解的预训练”，载于第58届计算语言学协会年会论文集，ACL 2020，线上，2020年7月5日至10日，D. Jurafsky、J. Chai、N. Schluter 和 J. R. Tetreault 编。计算语言学协会，2020年，第 8413–8426 页。[26] B. Bogin, J. Berant 和 M. Gardner，“利用图神经网络表示模式结构以进行文本到 SQL 解析”，载于第57届计算语言学协会年会论文集，ACL 2019，意大利佛罗伦萨，2019年7月28日至8月2日，第1 卷：长篇论文，A. Korhonen、D. R. Traum 和 L. M`arquez 编。计算语言学协会，2019年，第 4560–4565 页。[27] M. Pourreza 和 D. Rafiei，“DIN-SQL：通过自我纠正实现文本到 SQL 的分解式上下文学习”，《神经信息处理系统进展》，第 36 卷，第 36 339–36 348 页，2023年。[28] X. Xie, G. Xu, L. Zhao 和 R. Guo，“OpenSearch-SQL：利用动态少样本和一致性对齐增强文本到 SQL”，《ACM 数据管理会议录》，第 3 卷，第 3 期，第 1–24 页，2025年。[29] B. Li, Y. Luo, C. Chai, G. Li 和 N. Tang，“自然语言到 SQL 的黎明：我们准备好了吗？ [实验、分析与基准测试],”，《VLDB基金会会议录》，第 17 卷，第 11 期，第 3318–3331 页，2024年。[30] K. Chen, Y. Chen, N. Koudas 和 X. Yu，“通过自适应弃答实现可靠的文本到 SQL”，《ACM 数据管理会议录》，第 3 卷，第 1 期，第 69:1–69:30 页，2025年。[31] H. Zhang, R. Cao, L. Chen, H. Xu 和 K. Yu，“ACT-SQL：利用自动生成的思维链进行文本到 SQL 的上下文学习”，载于 EMNLP（研究成果）。计算语言学协会，2023年，第 3501–3532 页。[32] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini 和 A. Saberi，“CHESS：用于高效 SQL 合成的上下文利用”，arXiv 预印本 arXiv:2405.16755，2024年。[33] J. F¨urst, C. Kosten, F. Nooralahzadeh, Y. Zhang 和 K. Stockinger，“基于真实用户查询评估文本到 SQL 系统的数据模型鲁棒性”，arXiv 预印本 arXiv:2402.08349，2024年。[34] B. Wang, R. Shin, X. Liu, O. Polozov 和 M. Richardson，“RAT-SQL：面向文本到 SQL 解析器的关系感知模式编码与链接”，载于第58届计算语言学协会年会论文集，2020年，第 7567–7578 页。[35] X. Wu, J. Yang, L. Chai, G. Zhang, J. Liu, X. Du, D. Liang, D. Shu, X. Cheng, T. Sun 等，“TableBench：全面且复杂的表格问答基准测试”，载于 AAAI人工智能会议论文集，第 39 卷，第 24 期，2025年，第 25 497–25 506 页。[36] J. Zhang, Z. Shen, B. Srinivasan, S. Wang, H. Rangwala 和 G. Karypis，“NameGuess：表格数据的列名扩展”，载于2023年自然语言处理经验方法会议论文集，2023年，第 13 276–13 290 页。[37] H. Lee, S. Yang, H. Oh 和 M. Seo，“生成式多跳检索”，载于2022年自然语言处理经验方法会议论文集，2022年EMNLP。计算语言学协会，2022年，第 1417–1436 页。[38] X. Yu, P. Jian 和 C. Chen，“TableRAG：用于异构文档推理的检索增强生成框架”，arXiv 预印本 arXiv:2506.10380，2025年。[39] M. I. L. Balaka, D. Alexander, Q. Wang, Y. Gong, A. Krisnadhi 和 R. Castro Fernandez，“Pneuma：利用大语言模型实现端到端系统中的表格数据表示与检索”，《ACM 数据管理会议录》，第 3 卷，第 3 期，第 1–28 页，2025年。[40] A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin 和 C. Yu，“查找相关表格”，载于SIGMOD会议，第 10 卷，2012年，第 2 213 836–2 213 962 页。[41] K. Zhang, J. Zeng, F. Meng, Y. Wang, S. Sun, L. Bai, H. Shen 和 J. Zhou，“面向复杂问题回答的大语言模型推理树式问题分解”，载于

*AAAI*人工智能会议，第 38 卷，第 17 期，2024年，第 19 560–19 568 页。[42] Y. Peng, Q. Wang, L. Zhang, Y. Liu 和 Z. Mao，“问题链：一种面向复杂知识库问答的渐进式问题分解方法”，载于计算语言学协会 *ACL 2024* 研究成果。计算语言学协会，2024年，第 4763–4776 页。[43] S. Min, V. Zhong, L. Zettlemoyer 和 H. Hajishirzi，“通过问题分解和重新评分实现多跳阅读理解”，载于第*57*届计算语言学协会年会论文集，2019年，第 6097–6109 页。[44] X. Huang, S. Cheng, Y. Shu, Y. Bao 和 Y. Qu，“用于回答知识库复杂问题的问题分解树”，载于*AAAI*人工智能会议论文集，第 37 卷，第 11 期，2023年，第 12 924–12 932 页。[45]“斯坦福 Stanza 工具包”，https://stanfordnlp.github.io/stanza/。[46] Z. Lan 和 S. Li，“PS-SQL：利用预训练语言模型进行基于短语的模式链接，以实现文本到 SQL 解析”，载于*2024*年第六届自然语言处理国际会议（*ICNLP*）。IEEE，2024年，第 31–35 页。[47] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian 和 Z. Liu，“M3-embedding：通过自知识蒸馏实现多语言性、多功能性和多粒度文本嵌入”，载于计算语言学协会 *ACL 2024* 研究成果，2024年，第 2318–2335 页。[48] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazar´e, M. Lomeli, L. Hosseini 和 H. J´egou，“FAISS 库”，2024年。[49] K. Santhanam, O. Khattab, J. Saad-Falcon, C. Potts 和 M. Zaharia，“ColBERTv2：通过轻量级后期交互实现高效检索”，载于*2022*年计算语言学协会北美分会会议：人类语言技术论文集，2022年，第 3715–3734 页。[50] T. Ren, Y. Fan, Z. He, R. Huang, J. Dai, C. Huang, Y. Jing, K. Zhang, Y. Yang 和 X. S. Wang，“Purple：让大语言模型成为更出色的 SQL 编写器”，载于*2024*年 *IEEE* 第*40*届国际数据工程会议（*ICDE*）。IEEE，2024年，第 15–28 页。[51] X. Zhang, D. Wang, L. Dou, Q. Zhu 和 W. Che，“Murre：通过移除实现开放域文本到 SQL 的多跳表检索”，载于第*31*届计算语言学国际会议论文集，2025年，第 5789–5806 页。[52] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le 和 D. Zhou，“思维链提示能够激发大语言模型的推理能力”，神经信息处理系统进展，第 35 卷，第 24 824–24 837 页，2022年。[53] A. Khatiwada, R. Shraga 和 R. J. Miller，“数据湖表的模糊集成”，*arXiv* 预印本 *arXiv:2501.09211*，2025年。[54] S. Kweon, Y. Kwon, S. Cho, Y. Jo 和 E. Choi，“Open-Wikitable：用于开放域问答及表格复杂推理的数据集”，载于计算语言学协会 *ACL 2023* 研究成果，2023年，第 8285–8297 页。[55] T. Wang, S. Huang, Z. Bao, J. S. Culpepper, V. Dedeoglu 和 R. Arablouei，“数据集组装中的独特性最大化”，载于*2025*年 *ACM Web* 会议论文集（*WWW 2025*），悉尼，新南威尔士州，澳大利亚，*2025*年*4*月*28*日至*2025*年*5*月*2*日，G. Long、M. Blumestein、Y. Chang、L. Lewin-Eytan、Z. H. Huang 和 E. Yom-Tov 编。ACM，2025年，第 3219–3232 页。[56] F. Nargesian, K. Q. Pu, E. Zhu, B. Ghadiri Bashardoost 和 R. J. Miller，“组织数据湖以实现导航”，载于*2020*年 *ACM SIGMOD* 数据管理国际会议论文集，2020年，第 1939–1950 页。[57] T. Bleifuß, L. Bornemann, D. V. Kalashnikov, F. Naumann 和 D. Sri-vastava，“维基百科表格的秘密生活”，载于与第*47*届超大型数据库国际会议（*VLDB 2021*）同期举办的第*2*届异构数据存储搜索、探索与分析研讨会（*SEA-Data 2021*）论文集，哥本哈根，丹麦，*2021*年*8*月*20*日，CEUR 研讨会论文集系列，第 2929 卷。CEUR-WS.org，2021年，第 20–26 页。[58] E. Zhu, Y. He 和 S. Chaudhuri，“Auto-Join：利用转换连接表”，*VLDB* 基金会会议录，第 10 卷，第 10 期，第 1034–1045 页，2017年。[59] D. Ji, H. Luo, Z. Bao 和 J. S. Culpepper，“释放数据湖中的表集成潜力：成对可集成性判定、可集成集合发现及多元组冲突消解”，*VLDB* 杂志，第 34 卷，第 3 期，第 36 页，2025年。[60] X. Hu、C. Lei、X. Qin、A. Katsifodimos、C. Faloutsos 和 H. Rangwala，《Polyjoin：数据湖中的语义多键可连接表搜索》，载于计算语言学协会研究成果：*NAACL 2025*，2025 年，第 384–395 页。[61] F. Luo、H. Lan、H. Luo、Z. Bao、J. S. Culpepper、S. Sadiq 和 X. Wang，《释放表格数据湖中的缺失值填补：一种混合方法》，*VLDB* 杂志，第 35 卷，第 2 期，第 11 页，2026 年。[62] S. E. Robertson 和 H. Zaragoza，《概率相关性框架：BM25 及其后续发展》，信息检索基础与趋势，第 3 卷，第 4 期，第 333–389 页，2009 年。[63] D. Wang、L. Dou、X. Zhang、Q. Zhu 和 W. Che，《在可靠推理过程的引导下增强数值推理》，载于第*62*届计算语言学协会年会论文集（第*1*卷：长篇论文），2024 年，第 10 812–10 828 页。[64]《BGE-M3 模型》，https://bge-model.com/bge/bge m3.html。[65] R. Zhu、X. Liu、Z. Sun、Y. Wang 和 W. Hu，《缓解检索增强多跳问答中的检索丢失问题》，*arXiv* 预印本 *arXiv:2502.14245*，2025 年。[66]《源代码和基准数据集》，https://github.com/DataAutonomyLab/ DMRAL。[67] Y. Sun、H. Xin 和 L. Chen，《Reca：相关表增强的列语义类型标注框架》，*VLDB* 基金会会议录，第 16 卷，第 6 期，第 1319–1331 页，2023 年。[68]《并行计算》，https://en.wikipedia.org/wiki/Parallel computing。[69]《Bird-sql 排行榜》，https://bird-bench.github.io/。[70] Y. Zhang、J. Deriu、G. Katsogiannis-Meimarakis、C. Kosten、G. Koutrika 和 K. Stockinger，《ScienceBenchmark：用于评估自然语言转 SQL 系统的复杂真实世界基准测试》，*VLDB* 基金会会议录，第 17 卷，第 4 期，第 685–698 页，2023 年。[71] X. Zeng、P. Wang、Y. Mao、L. Chen、X. Liu 和 Y. Gao，《Multiem：高效且有效的无监督多表实体匹配》，载于*2024 IEEE* 第*40*届国际数据工程会议（*ICDE*）。IEEE，2024 年，第 3421–3434 页。

附录 A
元数据推断模块

由于不完整的元数据（即缺失列标题）可能会妨碍构建图时准确计算可并集性，我们引入一个元数据推断 模块，利用大语言模型推断并补全缺失的标题。一种简单的方法是从表中抽取几行（例如 10 行）作为完整上下文，并将其输入大语言模型以推断缺失标题。然而，这可能会纳入无关或相关性较弱的列，从而稀释上下文并降低有效性 [71]。为缓解这一问题，我们的模块采用受 [67]启发的定向方法。其核心思想是利用表内列分组，为大语言模型构建简洁且语义连贯的上下文；通过仅聚焦于小规模且有意义的相关列组，从而增强推断效果。具体而言，我们首先应用列语义发现算法 [6]，根据列值将每个表划分为语义连贯的列组。例如，给定一个包含以下列的表：C ={姓名、街道地址、城市、产品 ID、价格、销售日期}

，

算法可能会将它们划分为两个语义连贯的组： {姓名、街道地址、城市} ，用以表示“位置”这一语境，以及 {产品 ID、价格、销售日期}，归属于“交易”这一语境。然后，对于任何表头不完整的列，我们识别其对应的列组。接着，我们仅提取属于该组的列及其抽样行，以构建子表。随后，仅使用这一高度相关的子语境（即来自分区子表的抽样行）提示大语言模型进行表头推断。元数据推断提示如图 9 所示。

附录 B 提示

在本节中，我们补充本文使用的主要提示。问题分解提示（§III-B4）如图 10 所示。残差子问题生成提示（§IV-B）如图 12 所示。多步程序生成与执行引导的改进（§V-A）所用的提示分别如图 13 和图 14 所示。用于通过将非键列分组为列子集来准备评估数据集的提示（§VI-B）如图 11 所示。

附录 C
适配竞争方法的详情

JARUnion 和 MMQAUnion。我们在经过处理的元数据完整的表上应用其原有的表格检索策略，并通过纳入可并集性来扩充检索到的表。具体而言，对于每个检索到的表，我们将其与所构建图中同一簇的所有表合并，从而扩展该表，形成最终检索到的表。随后，我们在这些检索到的表上使用其方法来获取答案。

OpenSearch-Wild。我们通过增加以下内容将其适配到我们的设定中：(i) 在其预处理阶段进行数据库物化，该过程基于我们的推断元数据和表关系图，以及 (ii) 在其提取阶段基于已获得的必要列进行 Top-k 表格选择。具体而言，对于物化，我们将每个可合并表簇合并为一张关系表，并通过轻量级启发式方法构建主键-外键约束：我们选择唯一性比率最高的列作为主键；对于我们图中的每个可连接表对，我们选择与被引用的主键可连接性最高的列作为外键。对于选择，在获得 SQL生成所需的必要列之后，我们使用大语言模型从包含所提取列的表中选出 Top-k 候选表。

附录 D
表格检索和答案生成的错误分析

我们对 SpiderWild 和 BirdWild 中的所有失败案例进行错误分析，包括 (i) 检索错误 ，即至少漏掉一个金标准相关表；以及 (ii) 答案生成错误 ，即生成的程序得出了错误答案。下面总结主要错误模式及相应的改进方法。

检索。我们观察到三种主要的检索失败模式：(1) 缺少桥接表（*32%*）。有些问题需要借助中间桥接表连接两个表簇，但问题中并未提及该桥接表，因此检索时会遗漏（例如通过 disp 连接客户和银行卡）。(2)非信息性词语（*18%*）。有些词主要用于提供上下文，但几乎不能提供与表相关的线索，可能会干扰检索（例如，问题“在 SAT 考试阅读成绩平均分最高的学校中，其 5 至 17 岁学生的 FRPM 数量是多少？”中的“学校”）。(3)指代模糊（*50%*）。描述不充分的实体可能被错误地映射到表，导致真实证据表被遗漏（例如，“Harlan”可能指不同类型的实体，从而导致遗漏用户）。

为缓解这些错误，我们计划按以下方式改进连通表组构建（§IV-B）。对于缺少桥接表的情况，我们将把表关系图中 1–2 跳邻居扩展为候选表（作为桥接表候选项），然后重新执行组构建和基于覆盖率的重排序；对于非信息性词语，我们将进行后验证：如果排名靠前的组始终包含某个簇，且移除该簇后仍能保持连通性并获得相近的覆盖率，则将其移除，然后更新受影响的子问题；对于指代模糊的情况，我们将引入值级证据（例如，将实体与列值进行匹配），以验证候选表是否确实包含所提及的实体。

答案生成。我们观察到三类程序生成错误：（1）表选择错误（*27%*），即推理器选择了不正确的表，或遗漏了所需的证据表；（2）谓词值对齐错误（*27%*），即未能将自然语言值描述映射到正确的表格单元格值；（3）算子/结构错误（*47%*），即生成了不正确的算子或逻辑结构（例如，错误的比较运算符、缺失 DISTINCT，或不恰当的连接类型）。

为缓解这些错误，我们计划将三个标准的文本到SQL模块 [28]纳入思维链引导的多步生成与执行（§V-A）：（i）模式链接在子问题程序生成过程中，我们可以先将每个子问题对齐到候选列，并将表候选限制为包含这些列的表，从而减少表选择错误；（ii）值检索在谓词构造过程中，我们可以从所选列中检索一小部分候选单元格值（例如，对去重后的值进行模糊匹配），并将谓词值约束为这些候选值，从而减少谓词值对齐错误；（iii）自一致性投票在执行之前，我们可以生成多个候选程序并执行它们，然后选择最一致的执行结果，从而减少算子/结构错误。

附录 E 案例分析

我们对比了直接大语言模型 方法生成的子问题与 DMRAL 中与表对齐的问题分解器生成的子问题。表 X 展示了具有代表性的示例，突出了直接大语言模型 方法中常见的主要分解问题，以及 DMRAL 带来的改进。我们将这些问题分为三种类型：

关键信息缺失： 遗漏了完整明确问题所需的关键信息。例如，在第一个案例中，直接大语言模型 的分解遗漏了”账户开户”这一表述，而这对于正确识别相关账户表至关重要。相比之下，DMRAL 在第二个子问题中明确保留了这一信息（即完整性更好）。

冗余分解： 分解生成了相互重叠或重复的子问题。在第二个案例中，直接大语言模型 生成的两个子问题都冗余地提及了”超级英雄”，导致对超级英雄表的查询意图重复。相比之下，DMRAL 避免了此类冗余（即非冗余性更好）。

纠缠子问题： 这种分解会产生一个子问题，将多个不同的信息需求纠缠在同一个子问题中，从而增加回答的复杂性。在第三种情况下，直接大语言模型将“细分市场 *SME*”和“年份 *2013*”合并为一个子问题，尽管这两个元素不太可能在同一语境中同时出现。相比之下，DMRAL 将它们拆分为两个子问题（即表特异性更高）。

你是基于表格的数据理解领域的元数据推断专家。

任务： 你将获得一个包含部分缺失元数据的表，例如不完整或被 MASK 的表标题和列标题。你的目标是根据所提供的表格结构和表间上下文（即抽样行）推断并恢复缺失的元数据。

重要说明：
– 你必须在表标题，全样的“列标题”中提供的数量一致。– 不得添加、删除或重新排序任何列——只能使用你最合理的推断替换缺失项。– 保持原始标题顺序。

响应格式：

{ "updated_title": "...", "updated_标题": ["...", "...", ...]}

现在对提供的表进行元数据推断：
– [表标题] {表_标题}– [列标题] {标题}– [抽样行（随机抽取的 10 行）] {抽样_行}

图 9：元数据推断提示。

你是一位擅长基于表格的问答中多跳问题分解的专家。
任务： 将一个复杂问题分解为一系列更简单、最精简的子问题。每个子问题都必须符合以下准则：

– 使用名为“信息需求”的给定列表中每个条目归纳在一起的关键词组。– 保留原问题的完整含义和意图，不遗漏重要细节。– 确保子问题精简、不重复且自然。

响应格式：
{"子问题": [...]}

示例： [问题] 在 SAT 考试中，数学成绩平均分超过 560 分的学校中，有多少所学校是直接由特许学校资助的?
[信息需求] [['学校', 'SAT 考试', '特许学校资助'] , ['数学成绩超过 560']]
输出：
{"子问题": ["哪些学校的平均数学成绩超过 560？", "#1 中有多少所学校在 SAT 考试中直接由特许学校资助？"]}

现在将以下问题拆解：
– [问题] {问题}
– [信息需求] {表对齐的组}

图10：问题分解提示。

问题	问题	直接大语言模型	DMRAL
关键信息缺失	其中账户开立的，有多少出生于……的女性客户 *1950*年之前并一直居住在索科洛夫?	子问题1：哪些女性客户出生于1950年之前？子问题2：#1中有多少人留在索科洛夫?	子问题1：哪些女性客户出生于1950年之前并留在索科洛夫？子问题2：#1 有多少人开设了账户?
冗余分解	有多少超级英雄有棕色眼睛?	子问题1：哪些超级英雄有棕色眼睛？子问题2：有多少其中超级英雄来自 #1?	子问题1：哪些有棕色眼睛？子问题2：有多少超级英雄有 #1 吗?
纠缠子问题	平均值是多少月消费量细分市场中客户的 *2013* 年中小企业?	子问题1：哪些客户属于细分市场中小企业在年份2013? 子问题2：平均值是多少月消费量是?	子问题1：哪些客户属于细分市场中小企业？子问题2：是多少平均月消费量2013 年排名第一的是?

表 X：直接大语言模型 与 DMRAL 中观察到的分解问题示例。

您擅长通过将相关列分组为更小且具有明确意义的子表来整理表格。

任务： 你需要将一张杂乱的表重新整理，将其各列划分到多个子表中。

重组指南：

– 将自然相关的列分组。– 不要重命名、删除或调整每组中列的顺序。

回复格式：

```
{
  “表”:        [
    {
      “表        表标题":   "...",
      "列         标题":        ["..."    , "...", ...]
    },
    ...
  ]
}
```

现在根据提供的表格上下文重新组织表格。

– 表标题: {表_标题}– 列标题： {列_标题}

图 11：语义列分组提示。

你是一位擅长识别查询分解中语义缺口的专家。

任务： 给定用户问题和一组当前可用的表，找出完整回答问题所需但尚未提供的表，并将其表述为剩余子问题。如果已提供所有必需的表，则返回 None。

响应格式：

{"Residual Sub-question": ...}

示例：

[问题] 有多少名女性客户在 Jesenik 分行开设了账户?
[提供的表]金融客户表(client_id、性别、 出生日期、 地区_id)
金融处置表(disposition_id、客户_id、账户_id、类型)
输出：
{"剩余子问题": "Jesenik 分行的区名是什么？"}

[问题] 在含有碳元素的原子中，哪一个不含致癌化合物?
[提供的表]毒理学原子表(atom_id, molecule_id, 元素)
毒理学分子表(molecule_id, 标签)
输出：
{"Residual Sub-question": None}

现在对以下输入执行任务：

– [问题] {问题}– [提供的表] {
表}

图 12：残差子问题生成提示。

你 是结构化表格数据多表问答的 SQL 程序合成专家 .

任务： 你的目标是逐步进行推理，以生成正确的 SQL 程序。你将获得以下内容：– 一个自然语言问题。– 一组带有元数据的检索到的表。– 一组分解后的子问题（可能不准确或不完整）。– 可选的外部知识（例如问题短语与列之间的映射）。

推理过程：

– **步骤 1：** 仔细阅读问题和表元数据，以理解完整要求。– **步骤 2：** 评估分解后的子问题。如有需要，修改或重写这些子问题。– **步骤 3：** 对于每个修改后的子问题，逐步进行推理，编写相应的 SQL 查询。– **步骤 4：** 处理完所有子问题后，检查最后一个子问题的 SQL 是否完整回答了原始问题。

响应格式：

{ "推理": "...", "最终 SQL": "...
" }

示例：

[问题] 列出选修超过3门课程的女学生的姓名。
[检索到的 表] 选课记录(选课记录_id, 学生_id, 课程_id) 学生(学生_id, 姓名, 年龄, 性别)
[外部知识]"女性"指性别 ='F'
[分解后的 子问题] ['哪些学生选修了超过 3 门课程？', '#1 中女性学生的姓名是什么？']
输出：

```
{
   "推理": "让我们逐步思考。
   步骤 1：修订后的子问题 1：
   哪些学生选修了 3 门以上的课程?
   SQL: SELECT 学生_id FROM 选课记录 GROUP BY 学生_id HAVING COUNT(课程_id) > 3.

   步骤 2：修订后的子问题 2：#1 中女性学生的姓名是什么?
   SQL：SELECT DISTINCT name FROM student WHERE student_id IN (...) AND 性别 = 'F'."

   "最终 SQL": "SELECT DISTINCT name FROM student WHERE student_id IN (SELECT student_id
   FROM 选课记录 GROUP BY 学生_id HAVING COUNT(course_id) > 3) AND 性别 = 'F'"
}
```

现在对给定输入进行推理并生成 SQL 程序：

– [问题] {问题}– [检索到的表] {表}– [外部知识] {外部_知识}–
[分解后的子问题] {分解后的子问题}

图 13：思维链引导的程序生成提示。

Yo你是一名专家级 SQL 修正助手，负责处理结构化表格数据的多表问答 ta.

任务：给定一个自然语言问题、一组带有元数据的相关表格、一条因 SQL 执行错误而失败的错误 SQL 查询，以及可选的外部知识（例如问题短语与列之间的映射），你的任务是修正 SQL 查询，解决已识别的错误。

响应格式：

```
{
  "SQL": "..."
}
```

现在根据提供的上下文执行 SQL 修正：

– [问题] {question}– [表格] {tables}– [错误 SQL] {sql}– [SQL 错误] {sql_error}– [外部知识] {external_knowledge}

图 14：程序优化提示。