借助函数依赖图重排序器和大语言模型高效模式过滤扩展文本转SQL

Thanh Dat Hoang[1], Thanh Tam Nguyen[1], Thanh Trung Huynh[2],尹宏志[3], Quoc Viet Hung Nguyen[1]1格里菲斯大学（澳大利亚）， [2]VinUniversity（越南）， [3]昆士兰大学（澳大利亚）

摘要

大多数现代文本转SQL系统会将完整的数据库模式（主要是列信息）与用户的问题一同放入提示中，供大语言模型（LLM）处理。这种方法在小型数据库上效果显著，但面对超出大语言模型上下文长度限制的真实数据库模式时便会失效，即使是商业模型也不例外。近期的 Spider 2.0 基准测试充分体现了这一问题：其中包含数百张表和数万列，现有系统往往难以应对。目前的缓解方法要么依赖成本高昂的多步提示流程，要么独立地根据用户问题对列进行排序筛选，而忽略了列间结构。为扩展现有系统，我们提出 GRAST-SQL，这是一种开源且对大语言模型高效的模式过滤框架，可精简文本转SQL提示，具体包括：（i）使用融合值和元数据、具备查询感知能力的大语言模型编码器对列进行排序；（ii）通过一个基于函数依赖的轻量级图变换器，对彼此关联的列进行重排序；（iii）利用斯坦纳树启发式算法选择保持连通性的子模式。在真实数据集上的实验表明，GRAST-SQL 实现了近乎完美的召回率，且精确率高于 CodeS、SchemaExP、通义千问重排序器和嵌入检索器，同时保持亚秒级中位延迟，并可扩展至包含 23,000+ 列的数据库模式。源代码见 https://github.com/thanhdath/grast-sql。

1 引言

文本转SQL是将自然语言问题翻译为可执行SQL 查询的任务，也是数据库应用中长期存在的挑战 [30–32, 37]。尽管大语言模型（大语言模型）的最新进展带来了显著改进 [7, 19, 21, 26]，但由于用户查询日益复杂以及现代数据库模式的规模不断扩大 [23]，该任务仍然十分困难。当前大多数方法都会向大语言模型提供大量数据库模式信息——包括表名和列名、关系，甚至值——以及用户的问题，以生成SQL 查询 [17, 24]。

然而，Spider 2.0 [12] 等近期基准测试包含数百张表、数万个列以及大量元数据，暴露出其可扩展性问题。首先，纳入所有模式元素会增加提示长度，常常使其达到或超过大语言模型的上下文长度限制，并显著增加令牌成本 [12]。其次，这会引入噪声，使模型更容易关注无关的表或列，从而导致生成的SQL中出现错误的连接路径或属性选择 [12, 14]。

现有的扩展性缓解方案主要通过两种方式应对这些问题。第一种依赖拥有较大上下文窗口的商业模型，但由于单次扁平提示仍可能超出其限制，因此需要成本高昂的多步提示流程，部署费用也随之高昂 [12, 20, 21, 27]。例如，在 BIRD 开发集上采用 CHESS 式数据库模式筛选，每个请求会消耗 ~340K 个令牌（使用 GPT-4-turbo 时为 ≈ $3.4

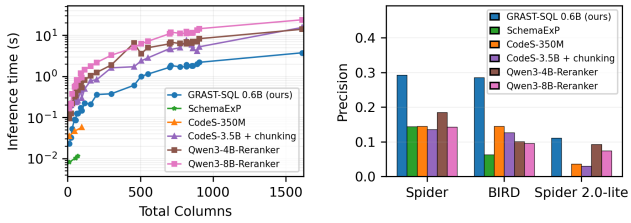

图 1：扩展性更强，效果更佳。左图：**BIRD 和 Spider 2.0-lite 上推理时间与列总数的关系。以往的排序器（如 SchemaExP、未进行分块的 CodeS）在列数超过 ~90-160 时便无法胜任，而 GRAST-SQL 则能平稳扩展。**右图：**GRAST-SQL 在 Spider、BIRD 和 Spider 2.0-lite 上的精确率始终更高，这得益于它对列语义的利用。**

），更不用说处理 1,000 个请求了 [5]。对于 Spider 2.0 这类复杂数据库，有些数据库甚至超过 2,000 万个令牌，因此需要 GPT-o3 [6] 等强大的推理模型。第二种方式是更经济的替代方案：采用结合关系感知排序的开源模式筛选（RESD-SQL、CodeS、SchemaExP、DCG-SQL、PURple），但仍受上下文长度限制，因而无法充分利用丰富的列语义，例如元数据和代表性值 [11, 14, 15, 22]。此外，人们可能会将列筛选视作文档检索在 RAG 流程中的应用，但朴素的 RAG 式嵌入和重排序器会忽略列间依赖关系，生成看似相关却遗漏关键字段的子集，导致 SQL 生成无法实现。这些问题在 Spider 2.0 和 BIRD 上尤为突出：宽模式常常会丢弃键列、保留噪声，并破坏可连接性。

为突破这些限制，我们提出了 GRAST-SQL：一套紧凑的开源流程，用于实现大语言模型高效模式过滤，无需依赖过于庞大的商业模型，同时能够扩展至海量真实数据库。GRAST-SQL 首先为每个数据库构建函数依赖（FD）图，并补充简明的元数据。给定一个问题后，它会（1）使用经过指令微调的大语言模型初始化查询感知列嵌入；（2）通过作用于函数依赖图上的关系感知图变换器对其进行优化；（3）应用斯坦纳树生成器，确保在 top-K 截断下保留有效连接。最终得到一个小型子模式，既保留了必要的连接结构和语义，又降低了令牌成本与噪声。如图 1（左）所示，现有的模式筛选器（如 CodeS、SchemaExP）无法处理宽模式：数据库包含数百列后，所需内容便会超出上下文窗口。相比之下，如图 1（右）所示，随着模式规模增长，我们的方法仍能保持稳定的延迟，并通过捕捉列语义而非仅依赖名称，获得更高的精确率。

我们的主要贡献总结如下。

表1：功能比较。最大列数* 表示不采用任何启发式方法（包括分块）时的原始容量，数值为近似值，并取决于模式分词（例如名称长度）；范围：单表/多表；Conn. 指连通性；关系建模表示列关系的学习方式。在我们的实验中，GRAST-SQL 在 Spider 2.0-lite 上可扩展至 23,067 列。

方法	范围	名称	含义	值	类型	连接	语言模型（架构）	关系建模	最大列数*（无启发式方法）
GRAST-SQL（我们的方法）	**多重**	✓	√（长）	✓	✓	✓	通义千问3（解码器）	图	**23,067**
SchemaExP	单一	✓	✗	✓	✓	✗	BERT（编码器）	语言模型注意力	≈ 90
RESDSQL	多重	✓	✗	✗	✗	✗	RoBERTa-L（编码器）	语言模型注意力	≈ 159
CodeS	多重	✓	√（短）	✗	✗	✗	RoBERTa-L/XXL（编码器）	语言模型注意力	≈ 117
PURple-SQL	多重	✓	√（短）	✗	✗	✓	RoBERTa-L（编码器）	语言模型注意力	≈ 117

- 我们提出了 GRAST-SQL，这是一种轻量级的大语言模型高效数据库模式过滤框架，集查询-列排序、基于图的重排序和连通性约束于一体，旨在使用紧凑型模型运行。
- 我们提出了一种函数依赖图，用于编码主键、外键和表内关系，并结合一种基于图的重排序器，在该结构中学习并传播列关系，从而提高列评分的准确性，同时确保连接路径有效。
- 我们证明，GRAST-SQL 的表现优于现有基线，能够扩展至包含数百张表和数千列的企业级数据库，在实现近乎完美召回率和更高精确率的同时，将中位延迟控制在一秒以内，并取得更优的 ROC/PR 曲线下面积；在 BIRD 上，文本转 SQL的提示长度最多可减少 50%。
- 我们发布了用于 Spider、BIRD 和 Spider 2.0-lite 模式筛选的训练集和评估数据集，以支持后续研究与比较。
- 我们将训练后的模型与完整代码库及脚本一并开源，支持在自定义数据库上复用，或基于新数据集重新训练。

下文将回顾相关工作（§2）、介绍我们的方法（§3）、报告实验结果（§5），并进行总结（§6）。

2 相关工作

文本转SQL。 现代文本转SQL技术始于 Syn-taxSQLNet [32] 和 RAT-SQL [29], 等神经解析器，它们对完整数据库模式进行编码；随后，PICARD [25]引入了约束解码。随着数据库模式规模不断扩大，筛选变得至关重要：RESDSQL 和 CodeS 根据问题相关性对元素排序 [14, 15], ，而 PURple-SQL 则通过连通性感知剪枝确保覆盖率。较新的系统，如 CHESS [27], 、CHASE-SQL [20], 和 Alpha-SQL [13] ，在多阶段流水线中运用提示、规划和重排序。在这些方法中，模式筛选仍是使用专有大语言模型 [11, 15],时减少噪声和令牌成本的关键；不过，许多方案依赖成本高昂的长上下文模型——这引发了对延迟、成本和数据隐私的担忧——或采用简单的启发式方法，容易遗漏关键列，从而降低复杂模式上的执行准确率。

我们的工作更进一步，提出了一种开源轻量级排序器，在大型数据库上保持文本转SQL准确率的同时，减少不必要的令牌成本。

模式筛选。 模式筛选仅选取与问题相关的表和列，从而缩短提示长度并减少噪声，同时保留可连接性。早期方法依赖模式/值链接和结构感知编码器：IRNet 将提及与模式元素关联起来 [8]；基于图的编码器对数据库结构进行建模 [1, 2]；RAT-SQL 应用关系感知注意力 [29]；BRIDGE 使用值增强字段 [18]；ShadowGNN 则提升了对噪声形式的链接效果 [4]。随后，RESDSQL 和 CodeS 使用 Transformer 对模式元素进行排序 [14, 15], ，DCG-SQL 通过模式图进行剪枝和优化 [11],，PURple 则通过 Steiner 树目标函数确保连通性 [22]。近期的智能体框架（如 CHESS 和 MAC-SQL）集成了模式选择器，但依赖反复调用大型专有大语言模型 [27, 28]。总体而言，模式筛选必须捕捉列关系——单独排序并不足够——然而，轻量级方法仍受上下文长度限制，往往会丢弃关键字段或保留过多噪声，且无法保证覆盖率或校准评分。

为克服这些局限，我们提出了一种适用于大语言模型的高效筛选器，能够处理长上下文中的列语义，同时捕捉列之间的关系，并可扩展至总计数千列。（参见表1中的功能比较。）

3 方法

3.1 问题形式化

文本转SQL中的模式过滤问题，着重于识别数据库模式的最小子集，以便将自然语言查询准确转换为可执行的 SQL 语句。令 $D = (T,\ C,\ F)$ 表示一个关系数据库，其中 T是表的集合，C是列的集合，F是外键关系的集合。给定查询 q，目标是生成一个 SQL 查询 s，其执行结果正确。模式筛选的作用是选出一个子模式（$T^\star,\ C^\star,\ F^\star$），其中仅包含回答 q所需的相关元素。这有两个目的：减少会误导 SQL 生成的噪声，并避免无关组件增加大语言模型的令牌成本。有效的模式筛选器必须在覆盖率和简洁性之间取得平衡，在保留必要模式组件的同时抑制无关内容。

示例*3.1.* 考虑一个数据库 D，其中包含 Students、Enrollments、Courses、Departments 等表。对于问题 q：“统计计算机科学 系开设的课程数量”，充分的子模式为 $T^\star$ = {Courses, Depart- ments}, $C^\star$ = {Courses.cid, Courses.dept_ id, Departments.did, De- partments.name}，外键集合为 $F^\star$ = {Courses.dept_id → Departments.did}。其他表与此无关，可以将其剪除，以减少噪声和令牌成本，同时保留正确的连接路径和完整的查询语义。对应的 SQL 为：

```
SELECT 计数(c.cid) AS num_courses
FROM Courses AS c
```

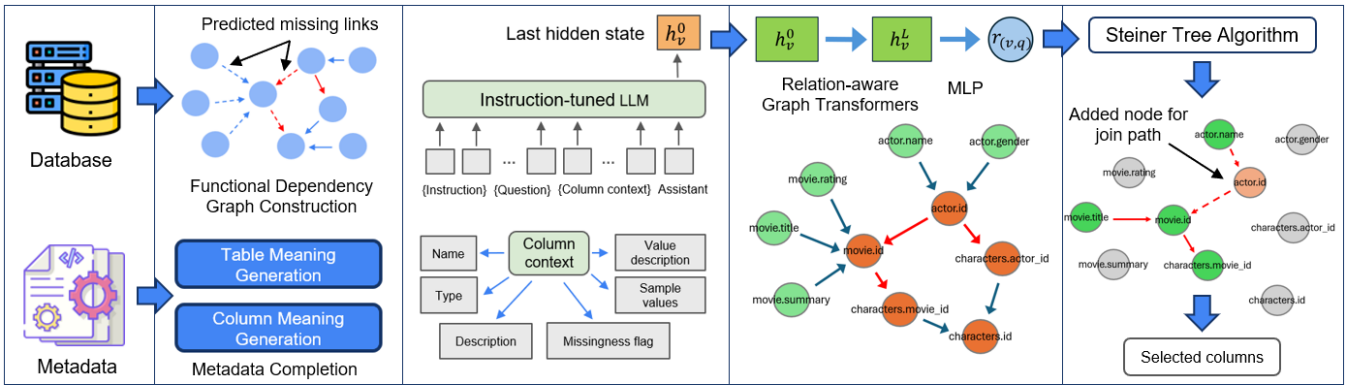

图 2：GRAST-SQL 概览。输入：数据库和元数据。模式增强器：构建函数依赖图并增强元数据。查询感知列编码器：通过指令微调大语言模型初始化列嵌入。基于图的重排序器：利用函数依赖图中的关系增强列嵌入。斯坦纳树扩展器：确保 Top-K 选择下的有效连接。

JOIN Departments AS d ON c.dept_id = d.did WHERE d.name = '计算机科学';

图 2 展示了我们用于模式筛选的四阶段方法 GRAST-SQL。各阶段将在后续章节中详细介绍。我们的方法不仅依赖启发式方法或长上下文大语言模型，而是结合查询感知排序、函数依赖图推理和斯坦纳树连通性。其实现颇具挑战，需要在不完整模式中恢复键、进行基于图的重排序、调整斯坦纳树，并针对规模化应用进行工程优化。

3.2 模式增强器

给定一个 SQL 数据库 D，模式增强器会构建 (1) 列上的函数依赖图，以及 (2) 增强元数据映射；两者均针对每个数据库构建一次，以供下游重复使用。

3.2.1 函数依赖图构建。 我们构建一个列级图 $G = (V, E)$，其中每个节点 $v \in V$ 对应一个列 $c \in C$，每条边 $e \in E$ 则表示一种模式依赖，该依赖可能由数据库声明，也可能由预测得出。构建此图的目的是确保结构上重要的列不会在筛选过程中被丢弃，即使它们与自然语言查询没有词汇或语义上的重合。如定理3.1所示，cid 和 did 等键列对于连接和聚合不可或缺，但与查询文本的表层形式毫无关联；要保留这些列，需要依靠结构连通性，而非纯粹的语义检索。

边类型。该图包含三种类型的依赖：

- 外键边：通过外键关系将源列连接到其引用列，定义见 F。
- 列到外键边：将表中的每个非键列连接到该表或被引用表中相应的外键列。
- 列到主键边：将表中的非主键列连接到该表的主键。

预测缺失键。在实践中，许多现实世界数据库（尤其是 BIRD 和 Spider 2.0 中的数据库）都省略了显式的主键或外键声明。为恢复这些缺失的结构，我们使用将模式元数据作为输入，通过 OpenAI 提示（GPT-4.1-mini）进行预测。这些预测仅用于丰富图，不会修改底层数据库。这种做法可行，因为许多键都遵循一致的命名约定，例如以 id 结尾的主键；而且，关系通常可以根据表之间复用的标识符推断出来，例如 locations.id → cards.location_id。

- 主键： 提示会接收一组表及其列名、类型和含义。任务是识别主键列表，并以 JSON 格式返回结果。随后，将预测的主键与任何已声明的键合并，为每个表生成 K_t。
- 外键： 使用相同的模式列表，提示要求模型以（$t_u.c_u \rightarrow t_v.c_v$）的形式返回可能的外键关系 JSON 列表。预测的链接$\hat{}$会与已声明的外键 F 合并，形成 F，其用途仅限于增强图的连通性。

3.2.2 元数据增强。 尽管结构连通性能够确保连接路径有效，并防止关键键列（例如主键和外键）被剪枝，但要识别哪些列与自然语言查询相关，仍需对表和列进行语义理解。为此，我们为每列补充简洁、易于理解的描述。

观测到的元数据。 本研究使用的所有数据集——Spider、BIRD 和 Spider 2.0——都包含外部元数据文件，其中为表和列添加了文本描述。对于已有的描述，我们直接采用，不作修改。

缺失的元数据。 在现实世界数据库中，尤其是 BIRD 和 Spider 2.0 中的数据库，许多表和列的描述缺失或经过缩写（例如 crs_cd、std_nm），导致语言模型难以将模式元素与自然语言问题正确对应。提供明确的含义有助于提高可解释性和检索质量。虽然这项任务理想情况下应由人工完成，但本研究使用 OpenAI 提示（GPT-4.1-mini）将其自动化。我们在两种情况下进行元数据增强：

- 表含义生成： 对于没有描述的表，我们向模型提供数据库和表名，以及列的列表（名称、类型、现有描述（如有），

以及样本值），要求模型用一句话概括该表的用途或内容。

- 列含义生成：对于缺少描述的列，输入包括数据库名、表名、列名、类型和样本值。模型会生成一句话来解释该列的含义。

我们仅对 BIRD 和 Spider 2.0 应用此流程，因为这两个数据集中缺失或含糊的模式元素很常见。此流程在 BIRD 开发集上总计消耗 22K 个 token（11 个数据库，$\approx$ \$0.009），在 Spider 2.0-lite 上消耗 32.6M 个 token （ 99 个数据库， $\approx$ \$13 ）。

3.3 问题感知列编码器

给定查询 q 和数据库模式 D，我们为每个$c \in C$ 构建列上下文，并将其转换为查询条件化嵌入，用于初始化函数依赖图中的节点特征（见图 2）。每个列上下文包括以下组成部分：

- 表名：列 c 所属的表 $t \in T$。• 列名：列 c的原始名称或标识符。
- 表描述：对表 t语义的文本说明。
- 列描述：对列 c语义的文本说明。
- 数据类型： c的 SQL 数据类型，例如整数、文本。
- 样本值：值检索器检索到的单元格值，所选条目与查询 q相匹配。
- 缺失标记：用于指示 c 是否可能包含缺失值或空值条目的布尔标记。
- 值描述：对存储在 c中的值的含义或编码方式的文本说明。

值检索器。 所有不同的单元格值都会预先离线编入稀疏倒排文件。运行时，系统会使用 BM25 为每个值 v 与查询的匹配程度打分，并将得分最高的k 个正向匹配项（$k = 2$）放入样本值 字段，因为与查询高度匹配的值能提供更有用的示例。如果没有找到匹配项，则使用一个具有代表性的示例；对于 Spider 2.0，我们跳过索引步骤，直接使用数据集提供的样本值。

嵌入提取。 为了获取列 c的查询感知嵌入，我们将查询 q和 c的上下文整理成指令遵循提示 I(q, c)。我们让大语言模型判断回答 q 是否需要 c （参见提示模板 1，§3.3）。我们不使用预测标签，而是提取模型用于为第一个答案词元（“yes”/“no”）打分的隐藏状态，该状态反映了模型对 c相关性的上下文评估。输入序列如下：

$$x_{q,c} = \langle \texttt{system} \rangle \, \mathcal{I}_{\text{sys}} \, \langle \texttt{end} \rangle \, \langle \texttt{user} \rangle \, \mathcal{I}(q, c) \, \langle \texttt{end} \rangle \, \langle \texttt{assistant} \rangle.$$

其中 I_{sys} 是系统提示。图节点的查询感知初始嵌入$h_{v_c}^{(0)}$ v_c 取自输入末尾位置的隐藏状态，即解码下一个词元“yes”/“no”之前的最终隐藏状态。该状态捕获了 q与列上下文 c之间的交互，用于初始化列节点特征，随后由基于图的模块进一步细化。

提示模板 1：列与查询的相关性

```
<|im_start|>system
判断是否有必要使用该列（文档）
编写 SQL 查询时，是否需要使用列（Document），
依据提供的查询和指令。答案只能是“yes”或“no”。
<|im_end|>
<|即_开始|>用户
<指令>：给定一个自然语言数据库问题
（查询）和一段列描述（文档），判断
该列是否可能是回答问题所必需的。
<查询>：{q}
<文档>：{context(c)}
<|im_end|>
<|im_开始|>助手
<思考>\n\n</思考>\n\n
```

3.4 基于图的重排序器

在阶段 §3.2.1 构建函数依赖图 $G = (V, E)$ 后，每个列节点 $v \in V$都使用 §3.3 中的查询条件化嵌入 $h_v^{(0)}$进行初始化。为注入结构上下文，我们将关系感知图变换器 [34] 应用于函数依赖图。设类型化边集为 $\tilde{E} \subseteq V \times V \times \mathrm{R}$，其中 R = {foreign_key，列 $\rightarrow$ foreign_key，列 $\rightarrow$ primary_key}。经过 ℓ 层后，每个图节点 v的表示通过关注其邻居进行更新：

$$h_v^{(\ell)} = W_{\text{self}}^{(\ell)} \, h_v^{(\ell-1)} + \sum_{(u,v,r) \in E} \alpha_r^{(\ell)}(u, v) \, W_r^{(\ell)} \, h_u^{(\ell-1)}, \qquad (1)$$

其中，W（ℓ）self 和 W（ℓ）r是可训练权重，而 $\alpha_r^{(\ell)}(u, v)$ 是特定关系的（缩放点积）注意力系数。边被视为有向边；在实践中，我们添加反向关系以实现双向传播。

经过最后一层后，精炼嵌入 $h_v^{(L)}$同时捕捉列的查询相关性及其在函数依赖图中的结构作用。线性投影将该嵌入映射为相关性分数 $r(v, q)$，对应于节点 v的列 c：

$$r(v, q) = w^\top h_v^{(L)} + b, \qquad (2)$$

其中， w和 b 是可学习参数。训练时采用基于间隔的对比损失，使真实 SQL 中使用的列排名高于不相关的列。随后将所得分数传递给阶段 4，以选出构成有效连接路径的连通列子集。

3.5 斯坦纳树生成器

尽管第 3 阶段排名靠前的列通常已经包含构成有效连接路径所需的外键，但在严格的 top-K cut 下无法保证连通性；本阶段是一个可选的附加步骤，它通过在函数依赖图 [9, 22] 上执行斯坦纳树过程，插入任何缺失的键列，从而明确保留有效的连接路径。

令 $R = \{v_1, \ldots\ldots, v_m\} \subseteq V$为相关性分数 $r(v, q)$ 最高的 m列。目标是找到一个最小的连通子图 $G^\star = (V^\star, E^\star)$，使所有终端节点 R都包含在其中，同时保持较低的总边成本：

$$\min_{V^\star, E^\star \subseteq G} \sum_{e \in E^\star} w_e \quad \text{s.t.} \quad R \subseteq V^\star, \; G^\star \text{ is connected.} \qquad (3)$$

当 e将终端节点连接到同一张表中的主键列或外键列时，边权重设为 $w_e = 0$；否则设为 $w_e = 1$，从而使解倾向于经济型连接。我们采用一种贪心

近似算法，每一步都添加能够减少连通分量数量的最廉价边；这种方法在实践中已足够有效，且相对于 $|E|$ 的运行时间接近线性。

通过此过程引入的辅助列汇总于 $C_{\text{aux}} = V^{\star} \setminus R$，最终筛选后的数据库模式为 $C^{\star} = R \cup C_{\text{aux}}$。由于斯坦纳树是在函数依赖图上构建的，因此有效 SQL 查询所需的每个连接键都会出现在 $C^{\star}$中，从而为下游解码器提供简洁且完全连通的提示。

4 模型训练

4.1 训练数据构建

我们使用 Spider 和 BIRD 基准测试的训练集划分构建了两个训练数据集。在原始数据集中，每个样例由一个自然语言问题 q 及其对应的 SQL 查询 y 组成。为生成列级监督：

- 我们提取金标准 SQL 查询 y中出现的列集合 $\mathrm{C}^{+}(q)$。
- 同一数据库模式中的所有其他列均视为负候选项，记作 $\mathrm{C}^{-}(q)$。

由于许多问题除了标注的查询外，还存在多个正确的 SQL 查询，我们使用大语言模型（例如 CodeS [15], CHASE-SQL [20]）生成替代查询，以补充监督信号：将温度设为 1.0，生成 20 个 SQL 候选项，再选出执行结果正确的查询。这些查询中的列的并集扩大了真实集合 $\mathrm{C}^{+}(q)$。每个训练实例表示为 $(q,\ \mathrm{C}^{+}(q),\ \mathrm{C}^{-}(q))$；过滤掉标签不匹配的实例后，最终训练集包含来自 BIRD 的 9,369 个样例和来自 Spider 的 8,395 个样例。

4.2 两阶段训练流程

步骤 1：大语言模型重排序器训练。我们微调一个仅解码器大语言模型重排序器，为每个列 c 在给定问题 q 的条件下计算相关性分数 $f(q,\ c)$。输入将列上下文（参见 §3.2）与 q 拼接在一起，并置于指令式提示中，将答案限定为“是”或“否”；$f(q,\ c)$ 定义为解码第一步中词元“是”对应的未归一化对数几率。具体而言，我们使用不同规格（0.6B、4B、8B）的 *Qwen3-Reranker* [35] 作为排序器，并采用 LoRA。我们将查询和段落的最大长度设为 4096 个词元，以支持长上下文。

训练目标是组级InfoNCE损失：

$$\mathcal{L}(q) = -\log \frac{\exp\big(f(q, c^{+})\big)}{\sum_{c \in \{c^{+}\} \cup \mathcal{N}(q)} \exp\big(f(q, c)\big)}, \tag{4}$$

其中，$c^{+} \in \mathrm{C}^{+}(q)$ 是金标准 SQL y 中的一个正列，$\mathrm{N}(q)$ 是采样得到的负样本。每个训练实例除正列外还配有七个负样本，并且我们进一步利用批内负样本来增加多样性。训练进行两个轮次，有效批量大小为 32，学习率为 2×10^{-4}。

步骤 2：图 Transformer 训练。训练完大语言模型排序器后，我们冻结其参数，并使用它为每个列 $c \in C$计算初始查询感知嵌入 $h_v^{(0)}$，如 §3.3 所述。这些嵌入被用作函数依赖图 $G = (V, E)$ 中的节点特征。随后，我们训练一个多层图

表2：Spider、BIRD 和 Spider 2.0-lite 的数据库模式宽度统计。每个数据集的中位数、第95百分位数和最大值均据此计算。

数据集	#DBs	表 (中位数/第95百分位数/最大值)	列 (中位数/第95百分位/最大值)
Spider（训练集）	146	4 / 14 / 26	20 / 66 / 352
Spider（开发集）	20	3 / 8 / 11	18 / 49 / 56
BIRD（训练集）	69	5 / 19 / 66	34 / 126 / 457
BIRD（开发集）	11	8 / 12 / 15	64 / 159 / 201
Spider 2.0-lite	99	12 / 133 / 381	228 / 3910 / 23067

变换器，如 §3.4 所述，并优化以下基于间隔的对比损失：

$$\mathcal{L}(q) = \sum_{c^{-} \in \mathcal{N}(q)} \max\big(0, \gamma - f'(q, c^{+}) + f'(q, c^{-})\big), \tag{5}$$

其中，$f_{\prime}(q,\ c)$ 表示基于图的模型为列 c 输出的最终分数。训练轮次为 40，批量大小为 32，学习率为 5×10^{-5}。

5 实验

本节旨在回答以下研究问题：

（RQ1）GRAST-SQL 的精确率和排序质量是否优于现有的模式筛选基线？（§5.2）（RQ2）函数依赖图和施泰纳闭包能带来哪些益处？（§5.3）（RQ3）GRAST-SQL 如何扩展到包含数百张表和数千列的超宽数据库模式？（§5.4）（RQ4）它对层深度和隐藏维度等超参数设置有多敏感？（§5.5）（RQ5）GRAST-SQL 对阈值的变化有多稳健？（§5.6）（RQ6）GRAST-SQL 能否在端到端文本转SQL任务中降低令牌成本，同时不损害执行准确率？（§5.7）

5.1 实验设置

数据集。我们在大型数据库模式语料库上进行评估：

- *Spider* [33] 是一个广泛使用的自然语言到SQL基准测试，包含覆盖 138 个领域的 200 个数据库。它提供了 8,659 个训练样例、1,024 个开发集样例以及一个测试集。
- *BIRD* [16] 包含来自 37 个领域的 95 个真实世界数据库，拥有 9,428 条训练数据、1,534 条开发数据，以及一个隐藏测试集。其数据库规模远大于 Spider（平均行数为 549K，而 Spider 为 2K），并且通常包含外部证据文本段。
- *Spider 2.0* [12] 是一项近期的基准测试，包含 632 个文本到 SQL 工作流问题，旨在反映真实的企业场景，其数据库模式规模远大于早期语料库，也更加复杂。我们使用部分公开的 Spider 2.0-lite，仅选取公开完整数据库模式信息的数据库来构建评估数据集。最终得到 233 个评估样本，涵盖多种 SQL 方言（BigQuery、Snowflake 和 SQLite）。数据库可能包含超过 1,000 列；SQL 查询 ~100 行，并且由于带时间戳的复制数据，最多可达 70K 列。为使评估可行，同时保留语义覆盖率，我们借鉴 Spider-Agent 的方法执行表分组步骤，在列级筛选之前合并结构相同的表 [12]。

表 3：Spider 开发集、BIRD 开发集和 Spider 2.0-lite 公开集上的模式筛选性能比较。

方法	Spider 开发集				BIRD 开发集				Spider 2.0-lite			
	ROC	PR	召回率	精确率	ROC	PR	召回率	精确率	ROC	PR	召回率	精确率
模式筛选基线												
SchemaExP	0.945	0.778	0.994	0.144	0.334	0.043	0.992	0.063	-	-	-	-
CodeS-350M	0.977	0.831	0.998	0.145	0.952	0.696	0.992	0.145	-	-	-	-
CodeS-350M +分块	0.983	0.847	0.998	0.151	0.960	0.725	0.991	0.146	0.898	0.138	0.978	0.040
CodeS-3.5B + 分块	0.983	0.868	0.997	0.136	0.972	0.755	0.985	0.126	0.912	0.143	0.901	0.030
bge-m3	0.836	0.427	0.959	0.145	0.799	0.346	0.991	0.095	0.744	0.041	0.991	0.078
Qwen3-Embedding-0.6B	0.841	0.453	0.998	0.159	0.70	0.148	0.991	0.109	0.681	0.035	0.991	0.081
bge-reranker-v2-m3	0.808	0.418	0.959	0.142	0.650	0.084	0.991	0.096	0.728	0.020	0.991	0.080
bge-reranker-v2-minicpm-layerwise	0.850	0.499	0.959	0.142	0.647	0.095	0.991	0.096	0.784	0.068	0.991	0.083
Qwen3-Reranker-0.6B	0.922	0.663	0.959	0.147	0.884	0.470	0.992	0.096	0.841	0.073	0.991	0.077
Qwen3-Reranker-4B	0.944	0.723	0.959	0.185	0.939	0.612	0.992	0.101	0.915	0.214	0.991	0.093
Qwen3-Reranker-8B	0.943	0.744	0.959	0.143	0.923	0.607	0.992	0.096	0.899	0.241	0.991	0.074
Ours												
GRAST-SQL 0.6B	0.981	0.899	**0.998**	0.293	0.978	0.777	**0.992**	0.285	0.937	0.236	**0.991**	0.111
GRAST-SQL 4B	0.987	0.921	**0.998**	0.395	0.986	0.837	**0.992**	0.344	0.956	0.255	**0.991**	**0.114**
GRAST-SQL 8B	**0.988**	**0.928**	**0.998**	**0.454**	**0.987**	**0.850**	**0.992**	**0.360**	**0.967**	**0.343**	**0.991**	0.113

表2汇总了所有数据集（包括我们构建的 Spider 2.0–lite 评估集）的数据库模式统计数据。表中报告了表数和列数的中位数、第95百分位数及最大值，显示所有语料库都包含数据库模式规模极大的数据库，凸显了对可扩展模式筛选的需求：既要保持较高的召回率，也要抑制无关字段。

基线方法。 我们将其与涵盖排序、图/剪枝、大语言模型排序和嵌入检索的模式过滤方法系列进行比较：

- SchemaExP（模式扩展与剪枝） [36]：一种模块化预处理流程，先扩展候选字段，再对数据库模式进行剪枝，以实现可处理的解码。我们将剪枝阶段用作过滤器。
- CodeS [15]：RESDSQL 的 5B 开源文本转SQL模型 [14]。我们使用它们的数据库模式筛选变体作为基于评分的排序器，评估每列和每张表的相关性。按照默认设置，CodeS 在 Spider 和 BIRD 上选取排名前 6 的表和排名前 10 的列。在 Spider 2.0-Lite 上，为了获得较高的召回率，我们选取排名前 40 的表和排名前 80 的列。“+ 分块”设置会将模式元素划分为多个批次，以适应令牌限制。
- BGE 重排序器 [3]：交叉编码器模型（v2-m3 和更轻量的 v2-miniCPM）会为查询–列对打分，并给出每列的概率，从而支持阈值筛选。
- Qwen3 重排序器 [35]：来自 Qwen3 系列的先进大语言模型重排序器，广泛应用于检索增强生成系统，可为查询–列相关性评分。
- 基于嵌入的方法：我们采用嵌入检索器（BGE-m3 [3] 和 Qwen3-Embedding-0.6B [35]），根据查询与列之间的嵌入相似度对列进行排序。

指标。 我们评估为每列分配相关性分数的评分方法，包括 CodeS、语言模型重排序器、嵌入模型以及我们的方法。我们报告 ROC AUC（用于比较方法的主要指标）和阈值扫描下的 PR AUC。我们还报告高召回率（接近 1.0）时的精确率，这反映了文本转SQL中既要保留所有相关列、又要抑制噪声的需求。请注意，召回率无需精确达到 1.0，因为有些问题存在多个能够正确执行的 SQL 查询，超出了标注范围。

5.2 模式筛选性能

表 3 报告了 Spider、BIRD 和 Spider 2.0-lite 上的列级结果。我们将模式筛选基线（SchemaExp、BGE 检索器和重排序器、CodeS、Qwen3 重排序器）与我们提出的 GRAST-SQL 进行比较。所有评分方法都为每列分配相关性分数，因此我们使用 ROC AUC 和阈值扫描下的 PR AUC 来评估排序质量。由于召回率对文本转SQL至关重要，遗漏必要的列通常会使查询无法求解。此外，我们还报告高召回率工作点下的精确率：选择一个使召回率接近 1。0 的阈值，然后测量由此得到的精确率。这凸显了各方法在不牺牲覆盖率的前提下抑制无关列的能力；同时也要注意，较高的召回率往往会降低精确率。

我们的方法始终保持较高的召回率（在 Spider 上为 0.998，在 BIRD 上为 0.992，在 Spider 2.0-lite 上为 0.991），同时其精确率显著高于嵌入检索器（BGE-m3）和交叉编码器重排序器（BGE v2-m3、v2-miniCPM）。与通义千问 Qwen3 重排序器和 CodeS 相比，GRAST-SQL 在这一高召回率工作点上也能取得更高的精确率，展现出更优的召回率与精确率平衡。此外，它在各数据集上均取得最佳 ROC AUC 和 PR AUC（例如，在 Spider 上 ROC AUC 为 0.988，在 BIRD 上为 0.987，在 Spider 2.0-lite 上为 0.967），证实了其在阈值变化下的稳健性。值得注意的是，我们的轻量版本 GRAST-SQL 0.6B 也取得了令人瞩目的结果：仅需更少的参数，便能保持较高的召回率和精确率，因此适用于资源受限的环境或需要处理大量列的场景。这些结果表明，我们的方法能够生成紧凑且在结构上足够完备的列集合，这对于下游 SQL 生成至关重要。

5.3 消融研究

在图 3 中，我们报告了 Spider 和 BIRD 上的 top–k 召回率与精确率，k 的取值范围为 2 到 20。我们比较了三种配置：(1) 使用 0.6B 模型的完整 GRAST-SQL 流程；(2) 不包含斯坦纳树扩展、但保留图变换器的变体；(3) 不包含函数依赖图的变体，即同时禁用图Transformer优化和斯坦纳算法。

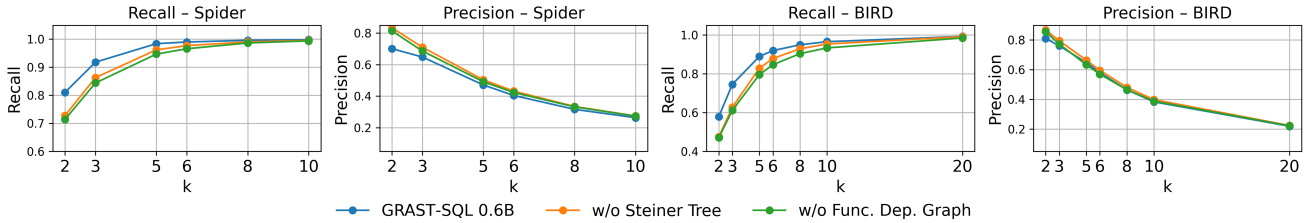

图 3：完整 GRAST-SQL 流程与两个精简变体在 Spider 和 BIRD 上的列级召回率与精确率。

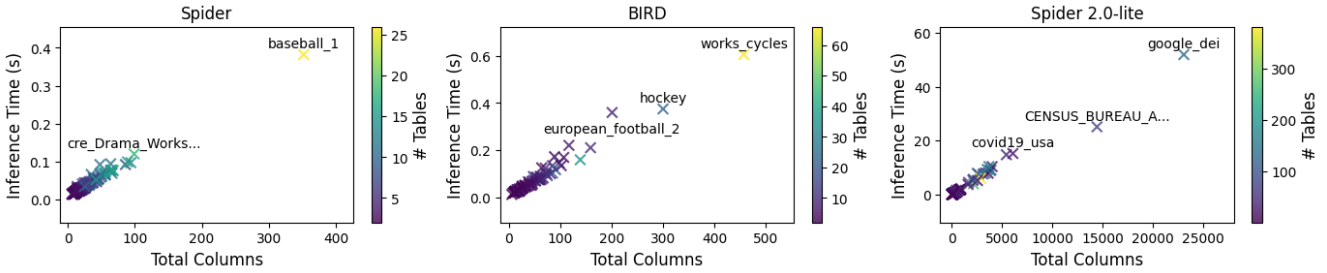

图 4：推理延迟与 。Spider、BIRD 和 Spider 2.0-Lite 上的数据库模式宽度（列数）；颜色表示表的数量 s.

我们的消融研究揭示了各组件的独特贡献。引入 Steiner 扩展显著提升了低 k 值下的召回率：在 k=2 时，Spider 上的召回率从 0.7272 提高到 0.8100，BIRD 上则从 0.4741 提高到 0.5790。正如预期，这也带来了适度的精确率折衷：在 k=2 时（例如，Spider 上的精确率从 0.8337 降至 0.7010；BIRD 上则从 0.8669 降至 0.8110），这是由于引入了额外的候选项。图建模至关重要：同时移除函数依赖图（及其图Transformer优化）和 Steiner 层（w/o Func. Dep. Graph），会导致 k=2 召回率降幅最大（Spider 上为 0.7137，BIRD 上为 0.4705），凸显了模式上下文的价值。仅靠大语言模型重排序器并不足够，w/o Func. Dep. Graph 与完整模型之间持续存在的差距便证明了这一点。总的来说，这些结果证实了 Steiner 扩展、图变换器和大语言模型各自的优势相辅相成，有助于实现强劲的整体性能。

5.4 效率评估

图 4 展示了模式筛选延迟与数据库规模之间的关系，数据库规模以表数和列总数衡量。本实验评估了我们的模型如何扩展到规模不断增大、复杂度不断提高的数据库模式。在本实验中，我们通过 vLLM [10]，（一种采用 KV 缓存机制的现代大语言模型推理引擎）部署模型，从而加快解码并降低延迟。这种兼容性也是我们方法的一项优势：不同于 CodeS 和 SchemaExP，我们的方法不需要特定任务的流水线。

在 Spider 上，大多数数据库的处理时间远低于 1 秒，中位延迟低于 0.2 秒。即使是 baseball_1 （26 张表、352 列）这样规模较大的案例，完成筛选也只需约 0.39 秒。在 BIRD 上，趋势类似：几乎所有数据库都能在毫秒级完成筛选，中位延迟低于 0.61 秒。最慢的案例 works_cycles（66 张表、457 列）也只需 0.61 秒。最后，在包含规模大得多的数据库模式的 Spider 2.0–Lite 公共数据集上，推理时间随规模平稳增长，规模扩大。对于 covid19_usa（21 张表、6066 列）和 CENSUS_BUREAU_ACS_2 （78 张表、14433 列）等数据库，完成筛选分别需要 15.22 秒和 25.09 秒。即使是规模最大的案例 google_dei（381 张表、23,067 列），也能在 51.86 秒内完成。这些结果表明，我们的方法在小型和中型数据库上具有实用的响应速度，同时能够扩展到超大型数据库模式；对于传统的基于上下文的方法而言，这类数据库模式难以处理。

5.5 敏感性分析

我们对两个关键超参数进行了网格搜索：图 Transformer 的层数（0–4）和隐藏维度大小（{256, 512, 1024, 2048}）。图 5、图 6 和图 7 展示了 BIRD、Spider 和 Spider 2.0-lite 数据集上的 ROC-AUC 和 PR-AUC 结果。

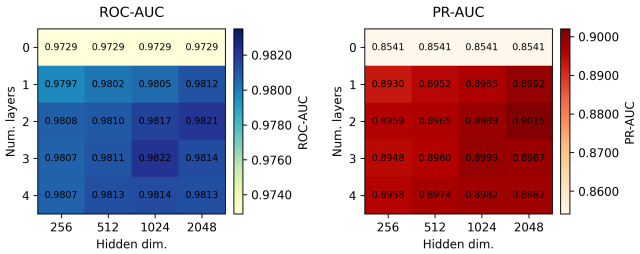

图 5：GRAST-SQL 0.6B 在 Spider 开发集上的 ROC/PR曲线下面积。

在 BIRD 上，性能随深度增加而提升，并在 3–4 层时趋于稳定；最佳 ROC-AUC（0.9793）出现在 3 层、隐藏维度为 2048 的配置下，而最高 PR-AUC（0.7873）则来自一个 4 层、维度为 1024 的模型。在 Spider 上，ROC-AUC 在 3 层、维度为 1024 时达到峰值 0.9822。在 Spider 2.0-lite 上，所有设置下的 ROC-AUC 均保持在 0.93 以上，最高值为 0.9396，出现在 3 层、隐藏维度为 2048 时；而 PR-AUC 的峰值为

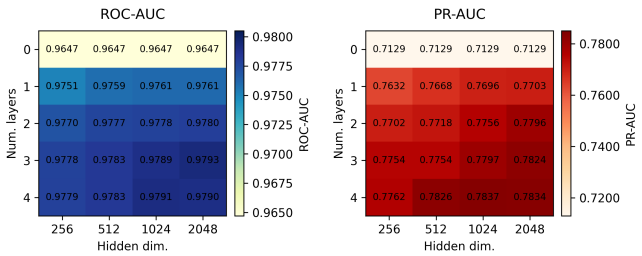

图 6：GRAST-SQL 0.6B 在 BIRD 开发集上的 ROC/PR。

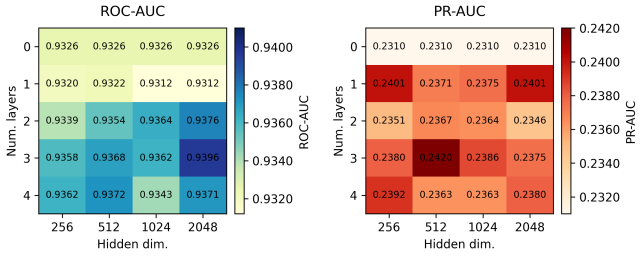

图 7：GRAST-SQL 0.6B 在 Spider 2.0-lite 上的 ROC/PR。

0.2420，出现在 3 层时。总体而言，这些结果表明，更深的模型能够持续提升 ROC-AUC，但最佳 PR-AUC 因数据集而异。在各项基准测试中，稳健的配置通常为 3–4 层，且隐藏维度至少为 1024，从而兼顾稳定性与性能。

5.6 阈值分析

我们通过扫描决策阈值，分析 Spider 和 BIRD 上逐列的得分，以计算宏精确率、召回率和 F_1,，其中得分高于阈值的列会被选中。指标既在原始选择结果上报告，也在应用 Steiner 步骤（恢复必要连接路径）后报告。如图 8 和图 9 所示，金标准与非金标准的分布区分明显，重叠很少，因此 ROC 和 PR AUC 较高。提高阈值会提升精确率、降低召回率；Steiner 步骤则会通过添加连接路径所必需的列，略微提高召回率，从而使高召回率区域的 F_1 更加稳定。不过，我们建议使用按最高K 或按最高百分比选择，而非固定阈值，因为最佳截断值可能因数据集而异，也可能在重新训练后发生变化。

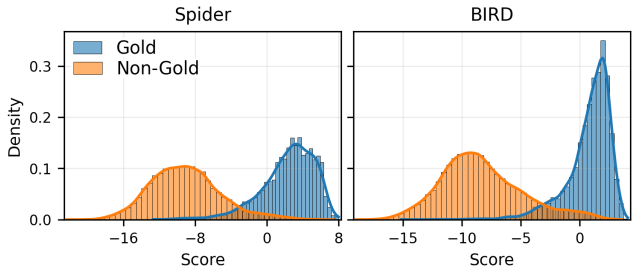

图8：金标准列与非金标准列的分数分布。

5.7 文本转SQL端到端比较

为评估 GRAST-SQL 对端到端文本转SQL性能和大语言模型推理成本的影响，我们在模型中对比了采用与

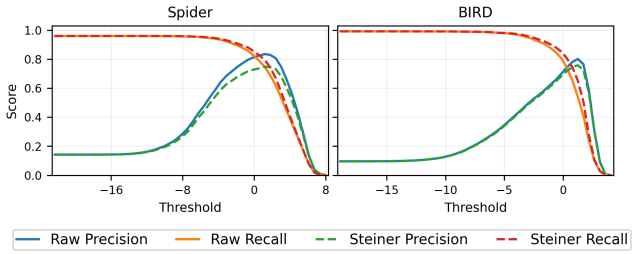

图9：宏精确率、召回率和 F_1 与阈值的关系。

未采用 GRAST-SQL 的情况，评测数据集为 Spider 和 BIRD 开发集。作为模式裁剪的预处理模块，GRAST-SQL 会分析问题和数据库模式，仅保留最相关的表和列，从而在将提示输入大语言模型之前缩短提示。这种精简降低了令牌消耗，进而减少 API 成本，且无需额外调用大语言模型。在实验中，我们使用 gpt-4.1-mini（一种成本效益较高、足以支持我们结论的大语言模型）将 GRAST-SQL 集成到 MAC-SQL 和 DIN-SQL 中。

表4：集成 GRAST-SQL 后的端到端性能。EX = 执行准确率，P = 提示令牌，R = 响应令牌，$\downarrow_P$ = 提示缩减（%）。

模型	Spider 开发集				BIRD 开发集			
	EX	P	R	$\downarrow_P$	EX	P	R	$\downarrow_P$
MAC-SQL	79.6	2.28M	88.6k	-	59.8	8.15M	1.03M	-
+ GRAST	80.3	1.59M	84.7k	30.2	59.7	4.07M	939.8k	50.0
DIN-SQL	79.0	9.30M	447.3k	-	58.9	38.7M	1.37M	-
+ GRAST	80.2	8.42M	469.5k	9.4	59.1	25.9M	1.34M	33.1

表4中的结果显示，在总体准确率不降低的情况下，令牌节省显著。在 BIRD 上，数据库模式规模较大，GRAST-SQL 将 MAC-SQL 的提示令牌减少了 50.0%（8.1M → 4.1M），将 DIN-SQL 的提示令牌减少了 33.1%（38.7M → 25.9M），准确率变化极小。在 Spider 上，MAC-SQL 的提示令牌减少了 30.2%，DIN-SQL 减少了 9.4%，且两者的准确率均略有提升。通过剪除无关上下文，GRAST-SQL 最多可减少 50% 的令牌，从而进一步降低与输入长度相关的运营成本，同时保持或提升性能。其轻量级设计确保与现有流程兼容，为高效部署文本转SQL提供了实用的即插即用方案。

6 结论

我们介绍了 GRAST-SQL，这是一种开源模式筛选管道，可对查询-列对进行排序、利用函数依赖，并应用斯坦纳树连通性，从而生成紧凑而完整的子模式。不同于依赖独立排序或高成本长上下文大语言模型的既有工作，我们的方法统一了大语言模型评分、图推理和连通性保证。其实现需要恢复缺失的键、设计基于图的重排序器，并针对 23K+ 列的模式规模调整 Steiner树算法以提高效率。在 Spider、BIRD 和 Spider 2.0-lite 上，GRAST-SQL 以较高精确率和近乎完美的召回率实现了最先进的 ROC AUC 和 PR AUC，在无需额外大语言模型调用的情况下，最多可减少 50 % 的提示令牌，并能在现代推理引擎上以亚秒级延迟运行。

参考文献

[1] Ben Bogin、Jonathan Berant 和 Matt Gardner。2019 年。使用图神经网络表示模式结构以进行文本转SQL解析。载于*ACL*。4560–4565。[2] Ben Bogin、Matt Gardner 和 Jonathan Berant。2019 年。面向文本转SQL解析的数据库结构全局推理。载于*EMNLP-IJCNLP*。3659–3664。[3] Jianlyu Chen、Shitao Xiao、Peitian Zhang、Kun Luo、Defu Lian 和 Zheng Liu。2024 年。M3-embedding：通过自知识蒸馏实现多语言、多功能、多粒度文本嵌入。载于*ACL*。2318–2335。[4] Zhi Chen、Lu Chen、Yanbin Zhao、Ruisheng Cao、Zihan Xu、Su Zhu 和 Kai Yu。2021 年。ShadowGNN：用于文本转SQL解析器的图投影神经网络。载于*NAACL-HLT*。5567–5577。[5] Yeounoh Chung、Gaurav Tarlok Kakkar、Yu Gan、Brenton Milne 和 Fatma Ozcan。2025 年。长上下文就是你所需要的一切吗？利用大语言模型的扩展上下文进行自然语言到SQL。*Proc. VLDB Endow.* 18, 8（2025），2735–2747。https://www.vldb.org/pvldb/vol18/p2735-ozcan.pdf[6] Minghang Deng、Ashwin Ramachandran、Canwen Xu、Lanxiang Hu、Zhewei Yao、Anupam Datta 和 Hao Zhang。2025 年。ReFoRCE：具备自我优化、共识约束和列探索能力的文本到SQL智能体。*arXiv preprint arXiv:2502.00675*（2025）。[7] Xuemei Dong、Chao Zhang、Yuhang Ge、Yuren Mao、Yunjun Gao、Jinshu Lin、Dongfang Lou 等。2023 年。C3：使用 ChatGPT 进行零样本文本到SQL。*arXiv preprint arXiv:2307.07306*（2023）。[8] Jiaqi Guo、Zecheng Zhan、Yan Gao、Yan Xiao、Jian-Guang Lou、Ting Liu 和 Dongmei Zhang。2019 年。迈向具有中间表示的跨领域数据库复杂文本转SQL。载于*ACL*。4524–4535。[9] Frank K Hwang 和 Dana S Richards。1992 22, 1 1992 55 89。斯坦纳树问题。*Networks*（），–。[10] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph Gonzalez、Hao Zhang 和 Ion Stoica。2023 年。使用分页注意力实现大语言模型服务的高效内存管理。载于*SOSP*。611–626。[11] Jihyung Lee、Jin-Seop Lee、Jaehoon Lee、YunSeok Choi 和 Jee-Hyong Lee。2025 年。DCG-SQL：利用深度上下文模式链接图增强文本到SQL的上下文学习。*arXiv preprint arXiv:2505.19956*（2025）。[12] Fangyu Lei、Jixuan Chen、Yuxiao Ye、Ruisheng Cao、Dongchan Shin、Hongjin SU、ZHAOQING SUO、Hongcheng Gao、Wenjing Hu、Pengcheng Yin 等。 [无日期]。Spider 2.0：在真实企业文本到SQL工作流上评估语言模型。载于*ICLR*。[13] Boyan Li、Jiayi Zhang、Ju Fan、Yanwei Xu、Chong Chen、Nan Tang 和 Yuyu Luo。2025 年。Alpha-SQL：使用蒙特卡洛树搜索进行零样本文本到SQL。载于*ICML*。[14] Haoyang Li、Jing Zhang、Cuiping Li 和 Hong Chen。2023 年。RESDSQL：解耦文本到SQL中的模式链接与骨架解析。载于*AAAI*，第 37 卷。13067–13075。[15] Haoyang Li、Jing Zhang、Hanbing Liu、Ju Fan、Xiaokang Zhang、Jun Zhu、Renjie Wei、Hongyan Pan、Cuiping Li 和 Hong Chen。2024 年。CodeS：构建用于文本到SQL的开源语言模型。*PACMMOD* 2, 3（2024），1–28。[16] Jinyang Li、Binyuan Hui、Ge Qu、Jiaxi Yang、Binhua Li、Bowen Li、Bailin Wang、Bowen Qin、Ruiying Geng、Nan Huo 等。2024 年。大语言模型已经能够充当数据库接口了吗？一个面向大规模数据库基础文本到SQL任务的大型基准测试。*NeurIPS* 36（2024）。[17] Raymond Li、Loubna Ben Allal、Yangtian Zi、Niklas Muennighoff 等。2023 年。Starcoder：愿源代码与你同在！*arXiv preprint arXiv:2305.06161*（2023）。[18] Xi Victoria Lin、Richard Socher 和 Caiming Xiong。2020 年。连接文本与表格数据：跨领域文本到SQL语义解析。载于*EMNLP*。4870–4888。

[19] Aiwei Liu、Xuming Hu、Lijie Wen 和 Philip S Yu。2023。对 ChatGPT 零样本 Text2SQL 能力的全面评估。*arXiv* 预印本 *arXiv:2303.13547* (2023)。[20] Mohammadreza Pourreza、Hailong Li、Ruoxi Sun、Yeounoh Chung、Shayan Talaei、Gaurav Tarlok Kakkar、Yu Gan、Amin Saberi、Fatma Ozcan 和 Sercan O Arik。2025。CHASE-SQL：文本到 SQL 中的多路径推理与偏好优化候选选择。发表于*ICLR*。[21] Mohammadreza Pourreza 和 Davood Rafiei。2023。Din-sql：通过自我纠错进行文本到 SQL 的分解式上下文学习。*NeurIPS* 36（2023），36339–36348。[22] Tonghui Ren、Yuankai Fan、Zhenying He、Ren Huang、Jiaqi Dai、Can Huang、Yinan Jing、Kai Zhang、Yifan Yang 和 X Sean Wang。2024。Purple：让大语言模型成为更出色的 SQL 编写器。发表于*ICDE*。IEEE，15–28。[23] Karlis Rokis 和 Marite Kirikova。2022。低代码/无代码软件开发的挑战：文献综述。发表于*BIR*。3–17。[24] Baptiste Roziere、Jonas Gehring、Fabian Gloeckle、Sten Sootla、Itai Gat 等。2023。Code llama：面向代码的开放基础模型。*arXiv* 预印本 *arXiv:2308.12950* (2023)。[25] Torsten Scholak、Nathan Schucher 和 Dzmitry Bahdanau。2021。PICARD：对语言模型进行受约束的自回归解码的增量解析。发表于*EMNLP*。9895–9901。[26] Ruoxi Sun、Sercan Ö Arik、Alex Muzio、Lesly Miculicich、Satya Gundabathula、Pengcheng Yin、Hanjun Dai、Hootan Nakhost、Rajarishi Sinha、Zifeng Wang 等。2023。Sql-palm：改进大语言模型对文本到 SQL 的适配（扩展版）。*arXiv* 预印本 *arXiv:2306.00739* (2023)。[27] Shayan Talaei、Mohammadreza Pourreza、Yu-Chen Chang、Azalia Mirhoseini 和 Amin Saberi。2024。Chess：通过上下文利用实现高效 SQL 合成。*arXiv* 预印本 *arXiv:2405.16755* (2024)。[28] Bing Wang、Changyu Ren、Jian Yang、Xinnian Liang、Jiaqi Bai、Linzheng Chai、Zhao Yan、Qian-Wen Zhang、Di Yin、Xing Sun 等。2025。MAC-SQL：面向 Text2SQL 的多智能体协作框架。发表于*COLING*。540–557。[29] Bailin Wang、Richard Shin、Xiaodong Liu、Oleksandr Polozov 和 Matthew Richardson。2020。RAT-SQL：面向 Text2SQL 解析器的关系感知模式编码与链接。发表于*ACL*。7567–7578。[30] Xiaojun Xu、Chang Liu 和 Dawn Song。2017。Sqlnet：无需强化学习即可从自然语言生成结构化查询。*arXiv* 预印本 *arXiv:1711.04436* (2017)。[31] Tao Yu、Zifan Li、Zilin Zhang、Rui Zhang 和 Dragomir Radev。2018。TypeSQL：基于知识的类型感知神经文本到 SQL 生成。发表于*NAACL-HLT*。588–594。[32] Tao Yu、Michihiro Yasunaga、Kai Yang、Rui Zhang、Dongxu Wang、Zifan Li 和 Dragomir Radev。2018。SyntaxSQLNet：用于复杂跨领域 Text2SQL 任务的语法树网络。发表于*EMNLP*。1653–1663。[33] Tao Yu、Rui Zhang、Kai Yang、Michihiro Yasunaga、Dongxu Wang、Zifan Li、James Ma、Irene Li、Qingning Yao、Shanelle Roman 等。2018。Spider：用于复杂跨领域语义解析和 Text2SQL 任务的大规模人工标注数据集。发表于*EMNLP*。3911–3921。[34] Seongjun Yun、Minbyul Jeong、Raehyun Kim、Jaewoo Kang 和 Hyunwoo J Kim。2019。图变换器网络。*NeurIPS* 32（2019）。[35] Yanzhao Zhang、Mingxin Li、Dingkun Long、Xin Zhang、Huan Lin、Baosong Yang、Pengjun Xie、An Yang、Dayiheng Liu、Junyang Lin、Fei Huang 和 Jingren Zhou。2025。Qwen3 Embedding：通过基础模型提升文本嵌入与重排序。*arXiv* 预印本 *arXiv:2506.05176* (2025)。[36] Chen Zhao、Yu Su、Adam Pauls 和 Emmanouil Antonios Platanios。2022。通过模式扩展弥合文本到 SQL 解析中的泛化差距。发表于*ACL*。5568–5578。[37] Victor Zhong、Caiming Xiong 和 Richard Socher。2017。Seq2sql：使用强化学习从自然语言生成结构化查询。*arXiv* 预印本 *arXiv:1709.00103* (2017)。