一种采用小语言模型和执行反馈的多智能体 Text2SQL 框架

Thanh Dat Hoang[1], Thanh Trung Huynh[2], Matthias Weidlich[3], Thanh Tam Nguyen[1], Tong Chen[4], Hongzhi Yin[4], Quoc Viet Hung Nguyen[1]1格里菲斯大学（澳大利亚）， [2]Vin大学（越南），[3]柏林洪堡大学（德国）， [4]昆士兰大学，澳大利亚

arXiv:2512.18622v1 [cs.DB] 21 Dec 2025

摘要

文本转SQL，即从自然语言文本生成 SQL 查询的任务，是数据工程中的一项关键挑战。近年来，大语言模型（LLM）凭借先进的理解和生成能力，在这项任务中展现出卓越的性能。然而，隐私和成本方面的考量使企业无法使用由外部大语言模型提供的文本转SQL服务解决方案。因此，企业转而采用可公开获取并可在内部托管的小型大语言模型（SLM）。但这些小型语言模型缺乏大型大语言模型的泛化能力，因而难以有效处理文本转SQL等复杂任务。为克服这些局限，我们提出了 MATS，这是一种专为小语言模型设计的新型文本转SQL框架。MATS 采用多智能体机制，为辅助智能体分配专门角色，从而减轻各智能体的工作负担并促进交互。该框架还采用基于强化学习的训练方案，利用执行过程中获得的反馈来协调这些智能体，使其即使在大语言模型规模有限的情况下也能保持有竞争力的性能。在基准数据集上的评估结果表明，MATS 部署在单台 GPU服务器上时，只需使用少得多的参数，便能达到与大型大语言模型相当的准确率。源代码和数据可在 https://github.com/thanhdath/mats-sql 获取。

CCS概念

• **信息系统** → **结构化查询语言**；**语言模型**； • **计算理论** → **多智能体学习**。

关键词

文本转SQL、小语言模型、多智能体系统

1 引言

文本转SQL，即将自然语言转换为 SQL 查询的任务，长期以来一直是研究领域面临的挑战 [49–51, 54]。用户查询和数据库模式日益复杂，加大了这项任务的难度 [39], 。尽管如此，基于大型版语言模型（大语言模型）的最新解决方案在文本转

2 [6, 23, SQL 33, 43] 2

方面取得了显著成果。文本转 SQL 方法有望让非专业人士通过自然语言界面查询数据库 [30, 45]。然而，现有的大多数解决方案都依赖作为服务提供的外部大语言模型 [28, 41], , 主要是 OpenAI GPT 的各种版本，用于生成 SQL 查询。这些方法将用户查询和模式表示与指令文本结合起来，以生成 SQL 查询 [22, 40]。然而，使用外部大语言模型服务也有其弊端。当数据库模式或查询日志等敏感数据被分享给第三方时，可能会引发隐私问题。平台，可能会违反保密性或使数据面临安全风险。此外，此类服务的持续费用可能造成沉重的经济负担，对小型组织而言尤其如此。

为了摆脱对外部大语言模型服务的依赖，近期的文本转SQL方法会使用指令数据对开源大语言模型进行微调。微调能够提升模型在特定任务上的性能，但需要大量计算资源和专业技术知识。此外，即便使用参数量高达150亿的模型，开源大语言模型所达到的准确率也远低于外部大语言模型服务 [16, 19]。为了寻求一种经济高效的解决方案，可以采用参数量通常较小（一般为1亿至50亿）的轻量级大语言模型（小型语言模型，SLM） [25]。这类模型经过优化，可在配备单个图形处理器的单节点服务器上运行。尽管使用小型语言模型提升了文本转SQL解决方案的效率，进而扩大了其适用范围，但模型效能方面的挑战依然存在。小型语言模型往往难以完成需要深入推理或理解复杂语境的任务，而文本转SQL正需要这类能力 [25]。其有限的容量使它们难以维持表或字段之间的关系，尤其是在模式庞大的情况下；它们也经常无法捕捉自然语言输入中的细微差别，从而导致SQL生成中的语法或语义错误。

本文沿用使用小型语言模型进行文本转SQL的思路，并提出 **多**智能体**架**构**文**本转**SQL**（MATS）框架，以将这一思路付诸实践。MATS 采用多智能体机制 [11, 13, 35]将文本转SQL任务分解为多个子任务，并由专门的智能体分别处理：模式调查智能体筛除无关的模式元素并检索相关列值；查询规划智能体逐步生成多个 SQL 查询；验证器利用数据库响应评估 SQL 输出；修复智能体根据验证反馈优化 SQL；流水线末端的选择智能体则从最终候选项中选出最佳 SQL 查询。为加强智能体之间的协作，我们设计了一种协作训练方案，称为执行反馈强化学习（RLEF）。与传统的基于人类反馈的强化学习不同，[29], RLEF 利用自动化数据库反馈生成多个响应，无需依赖成本高昂的人工标注数据。

MATS 实现的分而治之策略在效率和效果方面都大有裨益。由于智能体各司其职，专注于单一子任务，因此只需对小型语言模型的泛化能力进行高效管理，便足以实现较高的准确率。同时，通过强化学习整合智能体，使我们的框架能够有效处理复杂的用户查询和大规模数据集。此外，我们的框架便于适配开源小型语言模型，从而支持其在资源受限的设备和预算有限的场景中得到更广泛的应用。

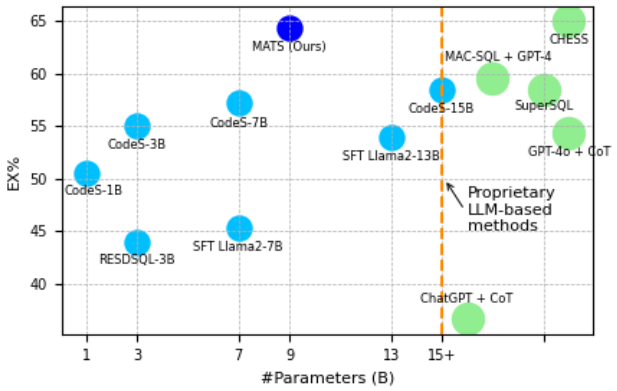


图 1：MATS 与其他方法在 BIRD 开发集上的执行准确率（EX%）。采用开源模型的方法与采用专有大语言模型的方法以虚线分隔。

我们将本文的贡献总结如下：

- 我们提出了一种新颖的多智能体框架，其中专门的智能体依靠小型语言模型协同解决文本转SQL任务。该框架定义了多个子任务，包括元素筛选、查询规划、查询结果验证、查询优化和查询选择。
- 我们提出了结合执行反馈的强化学习（RLEF）机制，使小型语言模型智能体能够在训练期间协作，从而显著提升其在文本转SQL任务中的表现。该机制依托偏好优化的最新进展 [12]，并基于一种采样方案来选取合适的响应。
- 我们通过人工标注、少样本提示和微调扩展 Spider 与 BIRD 数据集，构建了一个专为训练小型语言模型智能体而设计的综合数据集，以确保提供高质量样例，支持稳健学习。
- 我们通过全面的实验评估了 MATS，并观察到，尽管使用的模型规模显著更小，但其结果仍可媲美 GPT-4o + 思维链 [20] 和 CHESS [44], 等大型大语言模型。• 我们的源代码和数据可在 https://github. com/thanhdath/mats-sql 获取。

图 1 展示了我们实验中的主要发现：执行准确率与模型参数总量之间的关系。MATS 的模型总规模为 9B，专为资源受限环境优化。实验结果表明，MATS 能够在不牺牲性能的前提下实现高效推理，因此非常适合成本敏感型部署。

本文其余部分安排如下：第 2 节阐述所要解决的问题。第 3 节概述 MATS 框架的关键组成部分及其具体实现。第 4 节介绍我们提出的执行反馈强化学习（RLEF）方法。第 5 节展示评估结果；随后，第 6 节结合相关工作回顾我们的贡献，并在第 7 节总结全文。

2 模型与要求

我们首先刻画本文所要解决的问题（§2.1），然后详细阐述相关解决方案的要求（§2.2）。

2.1 问题表述

文本转SQL旨在完成这样一项任务：针对给定的自然语言问题 q，生成与之对应的 SQL 查询 Y。该查询基于数据库模式 S 构建，并可选择使用外部知识库 K。数据库模式 S由一组表 $\{T_1、T_2、\ldots\ldots、T_m\}$、一组列 $\{C_1、C_2、\ldots\ldots、C_n\}$以及一组外键关系 $\{R_1、R_2、\ldots\ldots、R_k\}$定义。可选的外部知识库 K 为模式提供上下文，有助于在存在歧义的情况下生成更准确的 SQL。

从数学角度来看，文本转SQL任务可表述为：

$$\mathcal{Y} = f(q, S, K \mid \theta),$$

其中，函数 $f(\cdot \mid \theta)$ 表示一个生成模型（例如神经网络），其可学习参数为 θ。

2.2 要求

我们认为，任何基于小型语言模型的文本转SQL解决方案都应满足以下要求：

(R1) 大型数据库模式。文本转SQL解决方案应能够处理大型数据库模式。当表和列数量过多，超出模型的上下文长度并削弱其理解能力时，处理起来便颇具挑战。真实场景中的模式通常包含含义重叠的列名和大量元数据，使相应任务更加复杂。例如，BIRD数据集中的数据库最多包含65张表和455列，增加了模式链接和 SQL 查询生成出错的可能性 [17]。

(R2) 含义模糊的列名和值。文本转SQL解决方案应能够处理列名和值的歧义，尤其是在多个列含义相似或值重叠时。例如，组织名称可能出现在不同的角色中，因此会对应数据库中的不同列。此时，将查询正确关联到相应列颇具挑战，也会增加 SQL 生成错误的风险。

(R3) 小型语言模型的推理能力较弱。基于小型语言模型的文本转SQL解决方案需要应对相关模型推理能力有限的问题。具体而言，能够提升大型模型推理能力的思维链提示，对小型语言模型的效果较差，甚至可能生成流畅却不合逻辑的推理结果 [47]。在 Text2SQL 任务中，这一局限尤为重要；已有观察发现，面对复杂的数据库模式，小型语言模型经常生成不准确的 SQL 查询 [17]。

（*R4*）指令遵循能力较弱。小型语言模型由于参数规模有限而难以遵循指令 [27], ，因此使用这些模型的文本到SQL解决方案需要考虑这一点。小型语言模型往往会对特定训练格式过拟合，而且通常缺乏对多样化指令微调数据集的接触，例如 In-FoBench [36] 或 IFEval [55]。因此，小型语言模型难以泛化到新的或不同类型的指令，也难以处理涉及依赖关系和顺序逻辑的指令。

3 MATS 框架

本节介绍我们的多智能体协作文本转SQL（MATS）框架。如图 2 所示，该框架采用多智能体协作范式，即将文本转SQL任务拆分为由单个智能体处理的子任务。

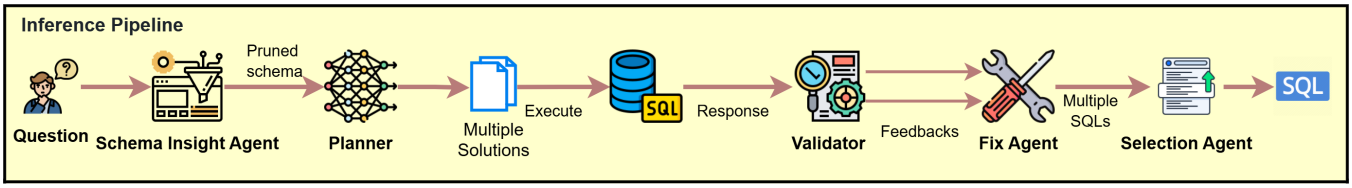


图 2：我们用于文本转SQL的多智能体框架概览。

给定用户查询和数据库模式，流程首先由**模式洞察智能体**开始，该智能体提取相关表和列，即使查询与存储值并不完全匹配也是如此。因此，该智能体明确应对了处理大型数据库模式（R1）以及含义模糊的列名和值（R2）的要求。接下来，**规划器智能体** 将推理过程分解为思维链。它通过生成 SQL 候选项来应对小型语言模型（SLM）的推理能力较弱问题（R3）。随后，**验证器智能体** 评估这些候选项及其执行结果，识别潜在错误。任何检测到的问题都会由**修复智能体**进行改进。最后，**选择智能体** 根据执行响应选择最佳 SQL 查询。

在本节余下部分中，我们将详细介绍各个单个智能体的实现（§3.1- §3.5），然后介绍用于微调的训练数据的构建（§3.6）；这些数据也旨在弥补小型语言模型推理能力较弱（R3）的问题。

3.1 模式洞察智能体

给定用户提出的问题作为输入，模式洞察智能体会筛除无关的模式元素，并检索相关列值。为此，它包含两个组件：模式筛选和值匹配。模式筛选会剔除不太可能有助于生成正确 SQL 查询的表和列。在我们的框架中，我们采用 CodeS [19] 来实现这一目的。它使用双向编码器对表和列进行排序，并舍弃与给定用户问题相关性较低的表和列。值匹配利用 BM-25 算法 [1] 识别与输入查询高度匹配的列值。此功能对于选择合适的列以生成准确的 SQL 查询至关重要。具体而言，给定输入问题 q，对于经模式筛选后得到的精简模式中的每一列 $c_i\{c_1$，c_2、……、$c_n\}$，我们检索一组候选值 $V_{c_i}=\{v_{i1}$，v_{i2}、……、$v_{im}\}$。我们计算问题与每个值之间的 BM25 相关性得分：

$$\text{Score}(q, v_{ij}) = \text{BM25}(q, v_{ij}).$$

然后，我们为每个列选取得分最高的 k 值：

$$V_{c_i}^{*} = \text{Top}_k\left(V_{c_i}, \text{Score}(q, V_{c_i})\right),$$

其中，通常 $k = 2$。如果没有值获得正的 BM25 分数，我们会从 V_{c_i}中选取一个有代表性的示例值。随后，选取的值 $V_{c_i}^{*}$会被纳入数据库模式提示，为模型进行 SQL 生成提供上下文线索。

3.2 规划器智能体

规划器智能体将推理过程分解为小型、逐步执行的操作，从而生成 SQL 查询，使小型语言模型能够构建准确且结构良好的查询。作为系统的核心组件，该智能体负责将用户问题转换为 SQL 查询，以完成给定任务。

我们基于少样本示例手动设计了一个包含三个步骤的推理过程：1）确定选择目标；2）分析 WHERE 子句的条件；3）确定 FROM 和 JOIN 子句所需的表。这种系统化方法引导模型逐步判断要选择什么、应用哪些条件，进而确定要使用哪些表。图 3 展示了思维过程的一个示例。

设 x 表示一个数据样本，其中包含一个问题、一个数据库模式，以及一个可选的外部知识库。规划器智能体按如下方式生成计划 p 和 SQL 查询 s：

$$p \leftarrow \pi_p(x) \tag{1}$$

$$s \leftarrow \pi_p(x, p) \tag{2}$$

在我们的框架中，规划器会生成 K 条 SQL 查询：一条使用贪心解码，$K-1$ 另一条使用温度为 T 的多项式解码：

$$S = \{s_1, s_2, \ldots, s_K\} \leftarrow \pi_p(x, p, \mathcal{T}). \tag{3}$$

Planner

<Database schema>
Question: What is the unabbreviated mailing street address of the school with the highest FRPM count for K-12 students?

Select:
- The question asks for the "unabbreviated mailing street address" of the school with the "highest FRPM count for K-12 students".
- ``unabbreviated mailing street address`` → `schools.mailstreet`
- ``highest FRPM count for K-12 students`` → Max of `frpm."frpm count (k-12)"`
- Should select: `schools.mailstreet`

Condition:
- ``for K-12 students`` refers to `frpm."frpm count (k-12)"`, so the condition is `frpm."frpm count (k-12)" IS NOT NULL`.

Tables to use: `frpm`, `schools`

Final SQL query:

```
SELECT schools.mailstreet
FROM frpm
INNER JOIN schools ON frpm.cdscode = schools.cdscode
WHERE frpm."frpm count (k-12)" IS NOT NULL
ORDER BY frpm."frpm count (k-12)" DESC
LIMIT 1;
```

图 3：规划器的思维过程示例。

3.3 验证器智能体

验证器智能体根据从数据库收到的响应，重新评估生成的 SQL 查询，以识别错误。因此，验证器不仅会验证规划器的推理，还会检测需要修改的问题，例如查询中的语法错误，或导致空响应的查询。

尽管验证器在验证选择目标、确定相关表以及分析选择条件的目标方面与规划器相似，但这种冗余至关重要，因为

大语言模型通常无法在首次尝试时完成任务 [26]。验证器利用查询评估的响应来检测差异、改进 SQL 查询，并在查询失败时提供有针对性的反馈。具体而言，我们采用了两个专门的验证器智能体。

验证器选择。 验证器会评估生成的 SQL 查询是否根据问题意图准确选择了正确的列。LLM 生成的 SQL 查询常常会出错，例如选择了错误或不必要的列、遗漏了必要的列，或列的排列顺序不正确。

我们通过为验证器选择步骤设计专门的思维过程来解决这些问题，如图 4 所示。首先，验证器会检查输入 SQL 查询中所选的列。接着，它会从输入问题中提取短语，并将其映射到需要选择的目标列。最后，验证器会将所选列与目标列进行比较，并标记潜在问题。

此验证器仅处理不包含某些操作的查询，例如最小值、最大值、计数、平均值、求和、除法或 CASE WHEN。之所以做出这一限制，是因为选择不同列的多个查询仍可能产生相同的正确结果，否则可能导致验证器误判。在验证过程结束时，验证器会判断该 SQL 查询是否正确。

验证器条件。 此验证器旨在识别 SQL 查询中与逻辑条件相关的错误。在 SQL 查询中，条件可用于 WHERE 子句或 SELECT 子句（例如，使用 CASE WHEN 或 IF 语句的查询）。

图 5 展示了条件验证机制背后的思维过程。首先，验证器从 SELECT 子句中提取条件并解读其含义。然后，它分析 WHERE 子句中的条件。在解读这两个条件后，验证器会评估执行响应。在大多数情况下，如果执行响应包含空值或空集，则很可能表示条件有误（例如，筛选了错误的值，或在筛选条件中使用了错误的列）。随后，验证器会找出条件中可能存在的错误，并提出修正建议。例如，它可能建议添加“列 A IS NOT NULL”等条件，以过滤掉空值结果，或修正被 LLM 误解的不匹配条件。最后，验证器会判断该 SQL 查询是否正确。

组合验证器。令 v_s，v_c 分别表示验证器选择和验证器条件的指令。验证器智能体 π_v 结合这些指令、输入 x、SQL 查询 s 及其对应的执行响应 er进行处理，并生成如下反馈信号：

$$f_s = \pi_v(v_s, x, s, er) \tag{4}$$

$$f_c = \pi_v(v_c, x, s, er) \tag{5}$$

此处，f_s，f_c 分别表示针对验证器选择和验证器条件的反馈。

3.4 修复智能体

修复智能体利用从验证器智能体获得的反馈来改进 SQL 查询。它会处理反馈，以调整并

Validator Selection

<Database schema>
Question: What are the elements for bond id TR001_10_11?
External knowledge: TR001_10_11 is the bond id;

SQL query: SELECT atom.atom_id FROM atom INNER JOIN connected ON atom.atom_id = connected.atom_id WHERE connected.bond_id = 'TR001_10_11';

Execution response:
atom_id
0 TR001_10
1 TR001_11

Feedback:
1. Based on the SQL query, the query selects: ['atom.atom_id']
2. The question asks for ['elements for bond id TR001_10_11'].
3. Based on the question, the query should select: [atom.element].
4. Compare 1. and 3., the SQL query selects the wrong column; it selects `atom.atom_id` instead of `atom.element`.
5. Conclude: incorrect.

图4：验证器选择反馈示例。

Validator Condition

<Database schema>
Question: In which country can I find the circuit with the highest altitude?
External knowledge: highest altitude refers to max(alt);

SQL query: SELECT circuits.country FROM circuits WHERE circuits.alt = (SELECT MAX(alt) FROM circuits);

Execution response:
Empty DataFrame
Columns: [country]
Index: []

Feedback:
1. `circuits.country` in `SELECT` retrieves the country from the `circuits` table.
2. `WHERE circuits.alt = (SELECT MAX(alt) FROM circuits)` filters for the highest-altitude circuit.
3. If `alt` contains only `NULL`, `MAX(alt)` returns `NULL`, making the condition invalid, causing an empty result.
Fix: Add `IS NOT NULL` to exclude `NULL` values before applying `MAX`.
Conclude: incorrect.

图5：验证器条件反馈示例。

改进 SQL 查询，力求确保最终输出不再出现错误。

修复智能体以初始输入 x、规划器生成的 SQL 查询 s 以及一组反馈信号为输入，生成修正后的 SQL 查询。修复智能体的这一过程π_f可表示为：

$$s_f = \pi_f\left(x, s, \{f \in \{f_s, f_c\} \mid f \text{ indicates an error}\}\right), \tag{6}$$

其中s_f是修正后的 SQL 查询。

3.5 选择智能体

选择智能体是一个小语言模型，它根据在数据库上评估这些 SQL 查询时获得的响应，从多个候选项中选出最佳 SQL 查询。

选择智能体以包含 SQL 查询列表 $S = \{s_1, s_2, \ldots\ldots, s_K\}$ 及其执行响应 $ER = \{er_1, er_2, \ldots\ldots, er_K\}$ 的提示作为输入，并输出最佳查询的索引（或指出没有正确的查询）。当 $K > k$ 时，智能体将S拆分成更小的子集（每个子集最多包含 k），选出每个子集中的最佳查询，并重复此过程，直到只剩下一个候选项。形式上，选择智能体 π_s 的功能可表示为：

$$s^* = \pi_s(S, ER), \tag{7}$$

其中 s^* 表示所选查询（及其索引）。

3.6 监督微调的数据生成

为缓解小型语言模型推理能力的局限，我们设计了一种生成训练数据以微调模型的方法。为此，我们为每项任务手动标注少量训练样例，使小型语言模型能够采用结构化的推理流程并减少错误。对于每项任务，我们基于 Spider 和 BIRD 数据集构建训练数据，具体内容将在评估部分详细讨论。由于这些数据集仅提供输入问题 x 和真实 SQL 查询 $\hat{s}$，我们对其进行了扩展，添加了额外标注：规划器的规划过程 p、针对不同验证类型的验证反馈 v_s，v_c，以及根据反馈修正后的 SQL 查询。数据创建过程分为以下三个结构化步骤：

1) 人工标注：我们为每项任务手动标注最多五个具有代表性的示例。这些高质量示例可作为参考，引导每个智能体执行细小、渐进的推理步骤，从而最大限度地减少错误传播。

2) 少样本提示：基于人工标注的示例，我们使用 OpenAI 的 GPT-4o-mini 通过少样本提示技术，为数据集的其余部分生成额外的训练样本。这种方法确保了对数据集中各种场景的广泛覆盖。

3) 小型语言模型微调：我们使用少样本提示阶段生成的提示-响应对对小型语言模型进行微调。微调过程侧重于优化模型输出中的补全部分。这里，我们采用监督微调损失，其计算方式如下：

$$\mathcal{L}_{\text{completion}} = -\sum_{t=C+1}^{\tau} \log P_\theta(y_t \mid y_{<t}, \chi), \tag{8}$$

其中，C 是标记提示结束位置的词元索引；τ 表示序列中的词元总数，包括提示词元和补全词元；y_t 表示补全部分中位置 t 处的目标词元；χ 表示提示词元；P_θ 是模型参数 θ 预测的词汇表上的概率分布。

4 基于执行反馈的强化学习

鉴于小型语言模型在推理（R3）和指令遵循（R4）方面的能力有限，我们提出一种基于执行反馈的强化学习（RLEF）方法，以进一步改进 MATS 框架中的智能体（图 6）。RLEF 面向智能体无法生成正确 SQL 查询的情况，并依靠查询评估所提供的反馈来识别问题并探索有效的修正方法。下文首先介绍适用于各类智能体的模型（§4.1），然后探讨用于训练它们的动作识别方法（§4.2）。最后，我们详细阐述实际的训练过程（§4.3）。

4.1 建模

给定用户问题和数据库模式，我们旨在生成最能满足其意图的 SQL 查询。我们将该任务形式化为一种目标增强型部分可观测马尔可夫决策过程：

$$M = (\mathcal{S}, A, T, R, G, O).$$

流程定义包含以下组件：

- S 是一个状态集合；
- $A \subset V^L$ 表示从语言模型词汇表 V 中采样的动作空间，其中 L 为生成文本的最大长度；
- T: S $\times A \to$ S 是转移函数；
- $G \subset V^N$ 表示目标空间；
- R: S $\times A \times G \to \mathbb{R}$ 表示目标条件奖励函数；并且
- O 是一组观测 $o \in O$。

请注意，奖励函数的设计取决于训练所使用的具体算法。例如，DPO [37]和 ORPO[12] 算法不需要显式的奖励模型，因为奖励本身是从语言模型中内在推导出来的 [37]。本研究采用 ORPO 算法，因此在下文中，我们使用“选定动作”和“拒绝动作”这两个术语，以与该算法的框架保持一致。

我们针对 MATS 框架中的不同类型智能体，将该过程具体化如下：

对于规划器，目标、观测和动作定义如下：

- G 是包含正确 SQL 查询的补全集合。• 观测 $o_p \in O_p$ 是输入样本 x。
- 选定动作是一个补全，其中包含与真实执行结果相符的正确 SQL 查询；拒绝动作则是导致语法错误或执行结果不一致的查询。

对于验证器，具体设定如下：

- G 是能够让修复智能体将 SQL 查询修改为正确版本的补全集合。
- 观测 $o_v \in O_V$ 是提供给验- 证器的输入提示，由 x、x 的执行响应以及验证指令 $v_i \in \{v_s, v_c\}$ 组成。
- 选定动作是能够正确识别问题并帮助修复智能体修改 SQL 查询的反馈。拒绝动作则是对修复智能体而言不正确或用处不足的反馈。

对于修复智能体，我们得到如下实现：

- G 是正确 SQL 查询的集合。
- 观测 $o_f \in O_F$ 由输入样本 x、执行响应以及一系列表明 SQL 不正确的反馈信号组成。
- 选定动作为与真实标注结果一致的 SQL 查询。拒绝动作则是会导致语法错误或偏离真实标注的查询。

4.2 动作采样

每个智能体的动作空间都非常大，规模为 $|V|^L$，因此，识别有效动作以训练智能体至关重要。然而，随机采样或仅依赖智能体探索动作空间，通常无法获得有效动作。因此，我们采用两种策略来生成动作：

- 多项式解码策略：我们使用温度参数 T，在小型语言模型生成文本的解码过程中引入可控的随机性。这有助于训练后的智能体探索更多有效动作。
- 高级智能体辅助：规模更大、训练更充分的语言模型能够为任务生成质量更高的操作。

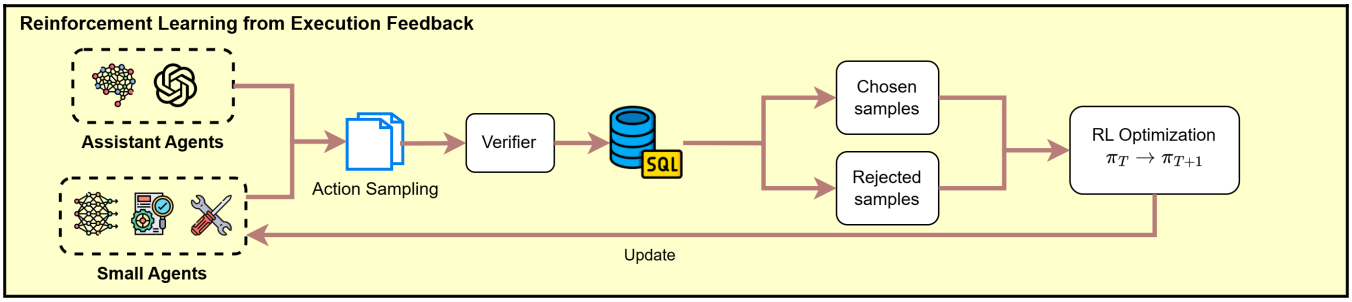


图 6：基于执行反馈的强化学习方法概览。

此外，我们引入了三个在训练期间利用 OpenAI GPT-4o-mini 的辅助智能体：

- 与推理阶段使用的较小型智能体相比，高级规划器智能体通常能够生成更准确的 SQL 查询。对于该智能体，我们采用少样本提示。
- 反馈编辑器智能体会修改反馈，以帮助修复智能体更有效地纠正 SQL 查询。
- 高级修复智能体能够以更高的成功率修复 SQL 查询。它会根据提供的反馈，通过提示来优化 SQL 查询。

对于每个智能体，在给定观测 o后，我们使用多项式解码策略以及相应的辅助智能体 A 生成一组动作。也就是说，我们生成一个大小为 K 的动作集 $A_P \subset A$，其中 $K-1$ 个动作是通过温度为 T 的多项式解码策略采样得到的，另一个动作则从辅助智能体 A 中采样得到。生成的动作集 A_P可能同时包含已选动作 C 和拒绝动作 R。随后，我们构造配对 P $=\{(o,\ c,\ r)\mid c \in$C $r \in$R$\}$。如果 A_P 仅包含已选动作 (C) 或仅包含拒绝动作 (R)，则忽略该训练样本。

算法 1使用 ORPO 对规划器进行迭代训练

输入：
1：训练数据集 D $=\{(x_i, \hat{s}_i) \mid i = 1, \ldots, N\}$。2：初始策略模型 π_P。3：高级规划器 $A_{\mathcal{P}}$、温度 T、迭代次数 T。输出：规划器的对齐策略 $\pi_{p,T}$。4：初始化$\pi_{p,0} \leftarrow \pi_P$5：对于 $i \leftarrow 1$ 到 T 执行 6：$D_p \leftarrow \emptyset$7：对于 $(x, \hat{s}) \in$ D 执行 8：真实 _ 响应 $\leftarrow$ 执行 $(\hat{s})$ 9：$A \leftarrow$ 采样 $(x, \pi_P, A_{\mathcal{P}}, T)$ 10：C $\leftarrow \emptyset$, R $\leftarrow \emptyset$ ▷ 初始化选中/拒绝动作 11：对于 $a \in A$ 执行 12：预测_响应 $\leftarrow$执行(a) 13：如果真实_响应 =预测_响应，则 14：C $\leftarrow$ C $\cup \{a\}$ 15：否则 16：R $\leftarrow$ R $\cup\{a\}$ 17：如果 $|$C$| > 0$，则 18：如果 $|$R$| = 0$，则 19：R $\leftarrow$ {""} ▷ 如果 R 为空，则添加空字符串 20：对于 $c \in$ C 中的每项执行 21：对于 $r \in$ R 中的每项执行 22：$D_p \leftarrow D_p \cup \{(x, c, r)\}$ 23：使用 ORPO 更新策略 π_P：$\pi_{p,i+1} \leftarrow \text{ORPO}(\pi_P, D_p)$ 24：返回$\pi_{p,T+1}$

4.3 智能体对齐训练

利用从采样策略中获得的选择-拒绝对，我们使用 ORPO [12]改进每个智能体。相比 PPO [42]或 DPO [42]等方法，我们选择 ORPO 是因为其流程更加简洁，能够简化训练并降低计算复杂度。通过将监督微调损失与基于优势比的惩罚项相结合，ORPO 在保留监督微调（SFT）所带来的准确率提升的同时，使模型输出与偏好保持一致。这有效避免了强化学习接续 SFT 时通常出现的灾难性遗忘问题。

ORPO 算法提出了一种单体式偏好对齐方法，无需参考模型和奖励模型，从而简化了偏好优化流程。该算法采用将监督微调损失与对数优势比惩罚项相结合的损失函数，使模型能够有效区分选择动作和拒绝动作。默认情况下，ORPO 会在整个提示词和补全上计算 SFT 损失。然而，我们修改了损失函数，使 SFT 损失仅应用于补全部分，从而让模型更加专注于生成高质量输出，而不是学习提示词的结构。

ORPO 的目标函数定义如下：

$$\mathcal{L}_{\text{ORPO}} = \mathbb{E}_{(x, y^w, y^l)} \left[\mathcal{L}_{\text{completion}} + \lambda \cdot \mathcal{L}_{\text{OR}} \right]$$

其中：

- Lcompletion 是标准的监督微调损失，仅在补全部分上计算，其实现方式为生成偏好响应y^w的负对数似然：

$$\mathcal{L}_{\text{completion}} = -\frac{1}{m} \sum_{t=1}^{m} \log P_\theta(y_t^w \mid x, y_{<t}^w)$$

- LOR 是对数优势比惩罚项，旨在促使模型提高生成偏好响应相对于非偏好响应的优势比：

$$\mathcal{L}_{\text{OR}} = -\log \sigma \left(\log \frac{\text{odds}_\theta(y^w \mid x)}{\text{odds}_\theta(y^l \mid x)} \right)$$

- $\text{odds}_\theta(y \mid x)$ 是生成序列 y的似然优势比，计算方式如下：

$$\text{odds}_\theta(y \mid x) = \frac{P_\theta(y \mid x)}{1 - P_\theta(y \mid x)}$$

- λ 是一个超参数，用于控制对数优势比惩罚的权重。

利用这一机制，我们从给定策略 π 开始，反复增强每个智能体。在每个训练阶段，策略 π_t 都会得到改进，从而产生策略 π_{t+1}。这一过程会循环进行，直到训练数据集的大小趋于稳定，这表明进一步微调已无法纠正错误。这些持续存在的错误通常与语义问题有关，而语义问题本质上与小语言模型的局限性密不可分。

规划器智能体的详细训练过程见算法 1。修复智能体的训练过程与之类似，不同之处在于，观测 x 包含生成的 SQL 查询以及来自验证器智能体的反馈信号。

算法 2 验证器智能体的动作选择过程

输入： 观测 o；动作集 A_s, A_c 根据观测 o 采样。2：真实 SQL $\hat{}s$；规划器 SQL 响应 s。

3：修复智能体 $\pi_{F,t-1}$。
输出： 选中的集合：C_s、C_c。拒绝的集合：$\mathcal{R}_s$、$\mathcal{R}_c$。
4：*true*_响应 $\leftarrow$执行($\hat{}s$)
5：规划器_响应 $\leftarrow$执行(s)
6：**如果** 规划器_响应 $=$*true*_响应 **那么**
7： **对于** $a \in \{A_s, A_c\}$ **执行**
8： **如果** a 表示 SQL 正确。**那么**
9： 将 a 添加到相应的选中集合 C
10： **else**
11： 将 a 添加到相应的拒绝集合 $\mathcal{R}$
12: **对于** $(a_s, a_c) \in (A_s, A_c)$ **执行**
13： $s_f \leftarrow \pi_{F,\ t-1}$（$o, s, \{a \mid a \in \{a_s, a_c\}$,
14： a 表示错误$\}$)
15： 预测_响应 $\leftarrow$执行(s_f)
16： **如果**预测_响应 $=$为真_响应 **那么**
17： 将 a_s 添加到 C_s，将 a_c 添加到 C_c
18： **else**
19: 将 a_s 添加到 $\mathcal{R}_s$，将 a_c 添加到 $\mathcal{R}_c$
20: **返回** C_s，C_c，$\mathcal{R}_s, \mathcal{R}_c$

验证器智能体的训练过程有所不同，因为它不会生成 SQL 查询来与真实查询进行比较。为了确定验证器智能体所选和拒绝的动作 $\pi_{V,\ t}$，我们纳入前一个对齐步骤中的修复智能体，$\pi_{F,\ t-1}$。如果验证器智能体生成的动作有助于修复智能体编写正确的 SQL 查询，则这些动作会被选中；否则，这些动作会被拒绝。此过程如算法 2 所述。

请注意，我们不处理规划器_响应 $\neq$为真_响应的情况，因为很难判断两种类型中哪一种反馈有误。例如，即使 SQL 查询不正确，选择反馈也可能是正确的，而错误完全在于条件反馈。

5 实证评估

在本节中，我们通过实验来回答以下研究问题：

（RQ1）在准确率和效率方面，MATS 是否优于基线方法？（RQ2）MATS 处理含噪和真实场景查询的鲁棒性如何？（RQ3）超参数会显著影响我们的模型吗？（RQ4）RLEF 如何提升 SQL 生成准确率？（RQ5）不同的强化学习算法会如何影响准确率？（RQ6）我们为小型智能体设计的思维过程重要吗？（RQ7）MATS 在不同领域中的表现如何？（RQ8）SQL 特征会如何影响 MATS？

5.1 实验设置

数据库。我们采用两个英文文本转SQL基准：Spider [52]和 BIRD [21]。

- *Spider* 是自然语言转 SQL 领域广受欢迎的基准数据集，包含 200 个数据库和多个表，涵盖 138 个不同领域。Spider 包含 7000 个训练样本、1024 个开发集样本，以及一个隐藏测试集。
- *Spider* 变体： Spider-DK [8], Spider-Syn [7], 以及 Spider-Realistic [4]通过引入同义词、缩写和经过修改的列名/值名称，为原始 Spider 数据集增添了真实世界的挑战。Dr.Spider [2] 在数据库、问题和 SQL 查询方面引入了 17 种扰动（模式/值同义词、排序顺序、数值变化），提供了更全面的鲁棒性测试。
- *BIRD* 包含 95 个数据库，总计 33.4GB，涵盖 37 个专业领域。BIRD 的训练集包含 9428 个样本，开发集包含 1534 个样本，另有一个隐藏测试集。BIRD 更具挑战性，其数据库平均每个约有 549K 行，而 Spider 的容量仅为 2K 行。除此之外，BIRD 还为特定样本提供证据，以帮助生成正确的 SQL 查询。

评估指标。 对于 Spider 基准，我们使用执行准确率（EX）进行衡量。它评估预测的 SQL 查询与真实 SQL 查询在数据库上是否产生相同的执行结果。然而，当错误的 SQL 查询偶然产生与真实标注相同的结果时，EX 有时会出现假阳性。为解决这一问题，Spider 基准还采用了测试套件准确率（TS） [19]。它评估预测的 SQL 查询能否通过在多个由自动增强生成的数据库实例上的执行测试，因此通过减少假阳性，使其成为一种更可靠的指标。

BIRD 基准主要采用执行准确率（EX）作为评估指标。此外，BIRD 还引入了有效效率得分（VES），用于评估正确生成的 SQL 查询的执行效率。与 EX 不同，在 VES 中，正确 SQL 查询的得分按真实 SQL 的执行时间除以预测 SQL 的执行时间计算。如果预测 SQL 的执行效率更高，其 VES 得分就会超过 EX。

基线方法. 我们将我们的方法与以下基线方法进行比较：

- 开源微调模型
  T5 [21] 是针对 Text2SQL 任务微调的基线模型，在不同规模（基础版、大型版、3B）下进行了测试。CodeS [19] 是一个开源模型（参数量为 1B–15B），通过在自然语言转SQL数据上持续预训练，使其专门适用于 Text2SQL。尽管 DeepSeek- R1 被视为开源模型且参数规模庞大（671B），我们并未将其纳入比较，因为在研究期间，由于流量过大，其 API 暂时无法使用。
- 基于闭源 LLM API 的思维链
  DIN-SQL [21] 通过思维链推理增强 SQL 生成，从而改进多步骤问题求解。DIN-SQL [34] 采用分解式上下文学习方法，将查询拆分为模式链接、查询分类和自我纠错等子任务。DAIL-SQL [9]整合

提示工程与系统化微调，根据问题与查询相似度以及均衡的组织策略优化示例选择。MAC-SQL [46] 采用多智能体框架，其中包含一个用于子问题处理的分解器。最后，CHESS [44] 利用模块化流水线，涵盖实体／上下文检索、模式选择和 SQL 生成，并借助大语言模型进行模式裁剪以最大限度地减少噪声。

设置。 模式洞察智能体使用 RoBERTa-large [24]（355M 个参数），而规划器和选择智能体则基于 LLaMA-3.2 3B 进行微调。验证器和修复智能体基于 LLaMA-3.2 1B 进行微调，使 MATS 的参数总量达到 9B。除模式洞察智能体外，所有智能体均采用 bfloat16精度和 FlashAttention [3]。

推理实验均在配备 24GB 显存 A5000 GPU 的机器上进行。模型通过 VLLM 框架 [15] 部署。监督微调时，我们将学习率设为 2.0×10^{-5}，批量大小设为 128，并对模型训练 4 个训练轮次。ORPO训练时，我们将学习率设为 5×10^{-6}，λ 设为 0.5，批量大小设为 64，训练轮次为 1，或训练不超过 800 步。对于规划器智能体，我们采样 $K = 10$ 个解，其中一个通过贪心解码生成，其余通过温度为 T = 1.0 的多项式采样生成。

5.2 端到端比较

表1和表2展示了MATS与其他基线方法的比较结果。在这些表中，CHESS$_{open-source}$ 指的是在其工作流程中使用Llama-3-70B和微调后的DeepSeek的CHESS方法。

准确率。 MATS在Spider和BIRD数据集上均表现出色。在Spider开发集上，MATS的执行准确率达到87.1，优于其他方法，包括CodeS-15B和MAC-SQL + GPT-4等更大型的模型。在BIRD开发集上，MATS模型的执行准确率达到64.73，VES%达到66.75。该表现与CHESS相当，后者集成了多个利用OpenAI的GPT-4-turbo的模块。总体而言，MATS在执行准确率和VES%方面均优于所有开源模型，同时与高度依赖专有模型（如OpenAI GPT）的闭源方法表现相当。

时间。我们仅与其他开源微调方法比较推理时间，因为闭源方法会将大语言模型部署在未知的物理设备上。我们注意到，MATS 的推理时间比 CodeS-15B 更长，这是由于（1）Llama 和 CodeS 在结构上存在差异；（2） MATS 中包含多个智能体。此外，MATS 在 Spider 和 BIRD 上的推理时间差异显著：在 Spider 上平均为 13.9 秒/样本，而在 BIRD 上为 22.03 秒/样本。这是因为 BIRD 包含许多大型数据库，导致执行时间更长。

模型复杂度。 自回归模型使用 VLLM 提供服务，其内存需求取决于模型规模和浮点精度，如表3所示。MATS 规划器需要 7.6 GB 的图形处理器内存，与 CodeS-3B 相当；而完整的 MATS 框架需要 21.2 GB，因此不同于 CodeS-15B，它可以部署在单个图形处理器上。尽管 MATS 框架的内存需求高于大多数其他开源

表1：MATS 在 BIRD 开发集上的评估。

方法	#参数	EX%	VES%	平均时间（秒/样本）
基于闭源 API 的方法				
ChatGPT + 思维链	-	36.64	42.30	
DIN-SQL +GPT-4	-	50.72	58.79	-
DAIL-SQL +GPT-4	-	54.76	56.08	-
MAC-SQL +GPT-4	-	59.59	66.39	-
GPT-4o + 思维链	-	54.43	58.73	-
CHESS + GPT-4	-	**65.00**	**66.69**	-
开源微调方法				
微调后的T5-3B 3B	23.34 25.57			-
监督微调 Llama2-7B	7B	45.37	46.98	-
监督微调 Llama2-13B	13B	53.91	58.77	-
CodeS-1B	1B	50.46	51.07	0.69
CodeS-3B	3B	55.02	56.54	1.06
CodeS-7B	7B	57.17	58.80	1.87
CodeS-15B	15B	58.47	59.87	3.52
CHESS $_{open-source}$	33B + 70B	59.86	-	-
Ours				
MATS	9B	**64.73**	**66.75**	22.03

表2：MATS 在 Spider 开发集上的评估。

方法	#参数	执行准确率	TS%	平均时间（秒/样本）
基于闭源 API 的方法				
GPT-4（少样本）	-	76.8	67.4	-
C3 + ChatGPT	-	81.8	71.4	-
DIN-SQL +GPT-4	-	82.8	74.2	-
DAIL-SQL + GPT-4	-	83.1	76.6	-
MAC-SQL + GPT-4	-	**86.75**	-	-
开源微调方法				
T5-3B + PICARD	3B	79.3	69.4	-
RESDSQL-3B	3B	84.1	73.5	-
监督微调 Llama2-7B	7B	77.8	73.5	-
监督微调 Llama2-13B	13B	81.6	76.6	-
CodeS-1B	1B	77.9	72.2	0.45
CodeS-3B	3B	83.4	78.1	0.71
CodeS-7B	7B	85.4	80.3	1.22
CodeS-15B	15B	84.9	79.4	2.38
Ours				
MATS	9B	**87.1**	**82.3**	13.9

表3：语言模型服务所需的GPU内存。

方法	浮点精度	GPU内存（Gb）
CodeS-1B	float16	2.4
CodeS-3B	float16	7.2
CodeS-7B	float16	16.8
CodeS-15B	float16	36
RESDSQL-3B[17]	float32	24.66
RESDSQL-3B + NatSQL[17]	float32	21.59
MATS 规划器	脑浮点数	7.6
MATS	脑浮点数	21.2

与其他方法相比，如表1和表2所示，该方法在执行准确率和推理效率方面表现更优。

表 4：在 Spider 变体上评估 MATS。

模型	Spider-Syn		Spider-Realistic		Spider-DK
	EX%	TS%	EX%	TS%	EX%
T5-3B + PICARD	69.8	61.8	71.4	61.7	62.5
REDSQL-3B[18]	76.9	66.8	81.9	70.1	66
ChatGPT	58.6	48.5	63.4	49.2	62.6
SQL-PaLM	70.9	66.4	77.4	73.2	67.5
CodeS-1B	66.5	59.3	70.9	61.8	64.7
CodeS-3B	75.7	69	79.9	74.4	71.8
CodeS-7B	76.9	70	82.9	77.2	72
CodeS-15B	77	69.4	83.1	75.6	70.7
MATS	**78.5**	**72.0**	**84.4**	**78.0**	**74.0**

5.3 鲁棒性基准测试上的评估

在本实验中，我们使用 Spider 变体和 Dr.Spider 数据集，评估 MATS 在含噪且更贴近现实的问题上的执行准确率。表4展示了 MATS 在 Spider-Syn、Spider-Realistic 和 Spider-DK 上的表现，凸显了其处理多样且复杂的查询变体的能力。

为进一步评估 MATS 的鲁棒性，我们在 Dr.Spider 上对其进行评估。请注意，该基准涵盖了针对问题、数据库模式和 SQL 查询的 17 种扰动，包括模式同义词替换、值同义词和缩写更新。然而，我们的模型仅在原始 Spider 数据集上训练，即未进行任何问题或数据库增强。

表5比较了 MATS 与其他基线方法的执行准确率。MATS 在数据库和 SQL 扰动方面优于所有基线方法。然而，在自然语言查询扰动方面，MATS 的准确率略低于 CodeS-15B（75.5%，而 CodeS-15B 为 76.3%）。这凸显了小型语言模型在理解自然语言方面的局限性，尤其是在处理同义词和语言变体时。总体而言，MATS 能够适应噪声较多的真实场景查询，表现出良好的鲁棒性，因此适用于现实世界中的 Text2SQL 任务。

5.4 超参数敏感性

候选数量。 由于规划器会生成多个响应，增加候选数量可能提高找到正确 SQL 查询的概率。我们通过以下方式探究选择效果：生成 K 条 SQL 查询以测量召回率，从而评估上限，并考察选择智能体识别正确查询的准确率。

图 7a 展示了随着候选数量增加，MATS 在 BIRD 开发集上的执行准确率。上限表示召回率，即当候选集中始终包含正确查询时可能达到的最高准确率。具体而言，如果 K 生成的候选中至少有一个包含正确的 SQL 查询，则该样本被视为正确。正如预期，增加 K 会提高召回率，从而提升上限。然而，MATS 的准确率并未始终呈现相同趋势。召回率在 $K = 30$ 达到 77.8%，而 MATS 的准确率仅为 62.78%，这凸显了潜在执行准确率与实际执行准确率之间的差距。原因可能是生成了更多错误的 SQL 查询，从而增加了选择智能体选中错误查询的可能性。

λ 以平衡 SFT 损失和对数优势比惩罚。 图 7b 展示了使用 RLEF 第 1 次迭代对规划器进行微调后，改变 λ 对 EX% 和语法错误数量的影响。在该图中，$\lambda = 0$ 表示不使用对数优势比惩罚，仅进行监督微调。引入 $\lambda > 0$ 并结合对数优势比后，准确率显著提升，并在 $\lambda = 0.25$ 达到峰值。同时，语法错误数量在 $\lambda > 0.2$ 显著减少。不过，我们的结果也表明，必须谨慎选择 λ，以平衡准确率和语法正确性。

候选生成的温度。 图 7c 报告了候选生成过程中不同温度设置下 EX% 的均值和标准差（std）。较低的温度对应较低的标准差，表明输出更加稳定。然而，由于生成的候选项多样性较低，找到正确解也会更加困难。相比之下，较高的温度会增加探索，使得在候选项中生成正确解更容易，但代价是波动性更大。需要注意的是，即使温度为 0.0，由于底层采样机制的影响，结果仍略有非确定性。

5.5 每轮 RLEF 迭代后的改进

图 8 展示了每轮 RLEF 迭代对规划器智能体和整个框架执行准确率的影响。对于规划器智能体，我们使用贪心解码报告准确率。结果表明，RLEF 显著提升了生成正确 SQL 查询的能力。在 BIRD 开发集上，RLEF 将规划器智能体的执行准确率从 53.65% 提高到 59.32%；在 Spider 开发集上，则从 83.3% 提高到 85.4 %。整个框架 MATS 也受益于 RLEF，并在多轮迭代中取得显著提升。在 Spider 开发集上，MATS 的执行准确率从初始的 85.5 % 提高到 87.1%；在 BIRD 开发集上，则从 59.06% 提高到峰值 64.73%，之后略趋于稳定。这些发现证实，执行反馈不仅有助于提升小语言模型的执行准确率，也是不可或缺的。

5.6 对齐算法的比较

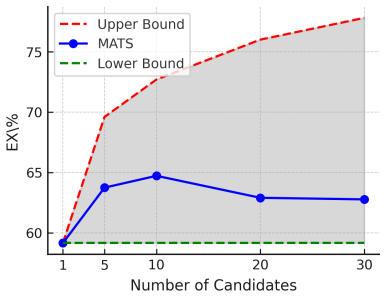
我们评估了 PPO、DPO 和 ORPO 等不同算法在对齐规划器策略方面的效果。其中，ORPO$_{original}$ 是 ORPO 的原始版本，ORPO$_{ours}$ 是我们修改后的 ORPO 版本，仅在补全部分上计算损失（见 §4）。对于 PPO，我们使用二元奖励：执行正确时奖励 1，否则奖励 0。对于 DPO，我们将 $\beta = 0.1$ 设置为学习率 5×10^{-6}。所有模型均训练一个训练轮次。

如表 6 所示，我们修改后的 ORPO 执行准确率最高（56.32%），优于 PPO 和 DPO。此外，我们对 ORPO 的修改表明，ORPO$_{ours}$ 消耗的 GPU内存更少，同时执行准确率更高。PPO 相较于监督微调基线略有提升，准确率从 53.65% 提高到 54.95%，但训练时间显著增加（每个训练轮次 12.2 小时）。相比之下，DPO 会导致灾难性遗忘，使准确率降至 44.98%。即使缓解了这一问题，其 GPU内存消耗（67 GB）和训练时间（1.8 小时）仍高于 ORPO。

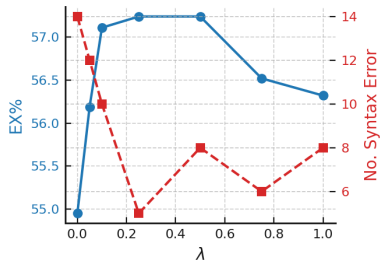
值得注意的是，ORPO 无需参考模型，因此 GPU内存用量几乎减少了一半。这使得批量大小可以更大，从而加快训练速度（每个训练轮次仅需 1.5 小时），同时保持

表5：MATS 在 Dr.Spider 上的评估。

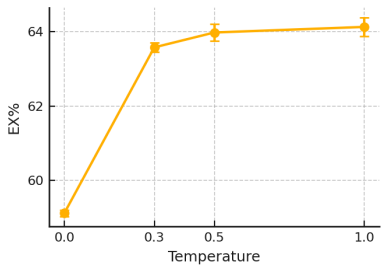
Type	扰动	#Sam.	REDSQL-3B [18]	ChatGPT +ZeroNL2SQL[10]	CodeS-7B	CodeS-15B	MATS
DB	模式同义词	2619	68.3	69.8	67.2	66.9	73.0
	模式缩写	2853	70.0	74.8	76.8	78.7	79.2
	数据库内容等价性	382	40.1	56.8	46.9	47.6	52.1
	平均值	-	59.4	67.1	63.6	64.4	**74.7**
NLQ	关键词同义词	953	72.4	74.0	73.0	73.5	73.1
	关键词载体	399	83.5	88.2	91.5	91.7	89.5
	列同义词	563	63.1	62.7	63.2	64.7	63.8
	列载体	579	63.9	71.7	80.7	79.1	75.3
	列属性	119	71.4	70.6	63.0	68.9	72.3
	列值	304	76.6	76.0	73.7	76.3	74.0
	值同义词	506	53.2	70.6	72.7	71.9	71.3
	多类型	1351	60.7	66.4	69.5	69.4	68.1
	其他	2819	79.0	79.4	81.5	81.2	81.2
	平均值	-	69.3	73.2	74.3	**76.3**	75.5
SQL	比较	178	82.0	73.6	77.5	71.9	77.5
	排序顺序	192	85.4	80.2	81.8	84.9	81.2
	非数据库数字	131	85.5	92.4	90.1	84.0	90.8
	数据库文本	911	74.3	80.7	80.5	80.7	79.3
	数据库数字	410	88.8	86.1	84.9	85.9	91.2
	平均值	-	83.2	82.6	83.0	81.5	**87.0**
All	全局平均值	-	71.7	74.9	75.0	75.1	**76.0**



(a) 随着候选数量增加，MATS 在 BIRD 开发集上的执行准确率。



(b) 微调后 BIRD 开发集上的执行准确率】【。规划器进行 RLEF 第 1 次迭代微调后，在改变 λ 时的执行准确率。



(c) 候选生成采用不同温度时，在 BIRD 开发集上的执行准确率。

图 7：影响 BIRD 开发集执行准确率的不同因素比较。

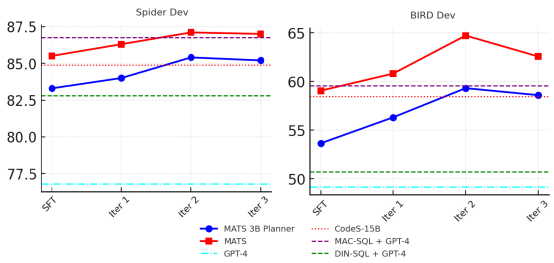


图 8：MATS 规划器在 BIRD 开发集和 Spider 开发集上使用贪心解码时，RLEF 每轮迭代的执行准确率。

与有限的 GPU 资源兼容，因此在资源受限的环境中是更实用的选择。

表 6：第 1 轮迭代后，不同对齐训练方法在 MATS 规划器模型上的 EX

模型	EX%	GPU内存 (Gb)	训练时间 （小时/训练轮次）
MATS 规划器（仅监督微调）	53.65	-	-
+ PPO	54.95	**46**	12.2
+ DPO	44.98	67	1.8
+ ORPO*original*	55.02	56	1.6
+ ORPO *ours*	**56.32**	52	**1.5**

5.7 规划器提示比较

在本实验中，我们比较了用于对规划器进行监督微调的不同提示策略。监督微调的输入包括提示及其对应的补全。我们构建了三个采用不同提示策略的监督微调数据集：（1）无思维，其中直接将 BIRD 训练数据集中的真实 SQL

查询用作补全；（2）思维链，其中使用 GPT-4o-mini 进行思维链提示来生成 SQL 查询，并将其用于微调；（3）少样本思维，其中采用我们在 §3.6 中介绍的数据生成机制。

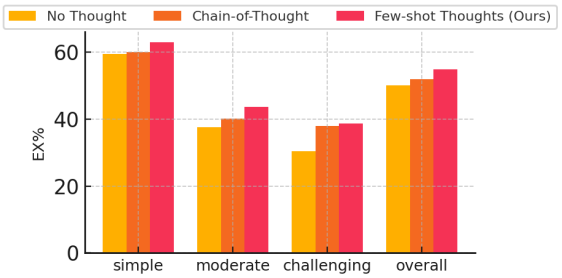


图 9：BIRD 开发集上的规划器提示对比。

图 9 的结果表明，少样本思维（3）始终优于思维链和无思维。在简单查询中，加入基于思维的提示仅带来小幅提升：执行准确率从 59.46%（1）提高到 60.11%（2）和 63.03%（3）。然而，对于更复杂的查询，其影响更为显著。在中等难度查询中，少样本思维达到 43.75%，比无思维高出 6.25%。在具有挑战性的查询中，少样本思维（3）达到 38.62%，而思维链为 37.93%，无思维为 30.34%，这凸显了结构良好的思维过程对 SQL 生成的重要性。

5.8 数据库领域适配

如图 10 所示，我们的方法 MATS 在 Spider 基准的各数据库领域中取得了最佳的总体执行准确率。尽管 MATS 在竞赛和社会等领域的表现优于 DAILSQL(SC) 和 REDSQL-3B + NatSQL 等现有基线方法，但在地理等某些领域的表现略低。这可能是因为 DAIL-SQL 和 CodeS-7B、CodeS-15B 使用了更大的模型，在地理领域具备更丰富的背景知识。尽管存在这些差异，MATS 在不同领域中仍表现出很强的泛化能力。

5.9 在 SQL 特征上的评估

Fig. 12 比较了 Spider 和 BIRD 数据集上不同 SQL 特征的执行准确率。MATS（我们的方法）始终优于 DAILSQL(SC)、CodeS-15B 和 REDSQL-3B 等基线方法，尤其是在不包含连接、排序或逻辑连接符的简单查询上。然而，子查询仍然是所有方法面临的一大挑战，与非嵌套查询相比，执行准确率下降了 10–30%。虽然 MATS 在这方面优于其他方法，但其性能仍有明显下降，这表明处理复杂 SQL 结构本身具有难度。

6 相关工作

虽然我们已经讨论了现有的文本转SQL方法，但这里我们将重点介绍与 MATS 底层技术相关的工作，即强化学习、偏好对齐和智能体协作方面的研究。

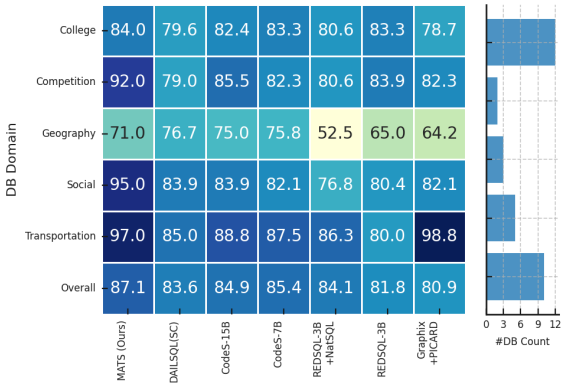


图 10：Spider 上不同领域的执行准确率比较。

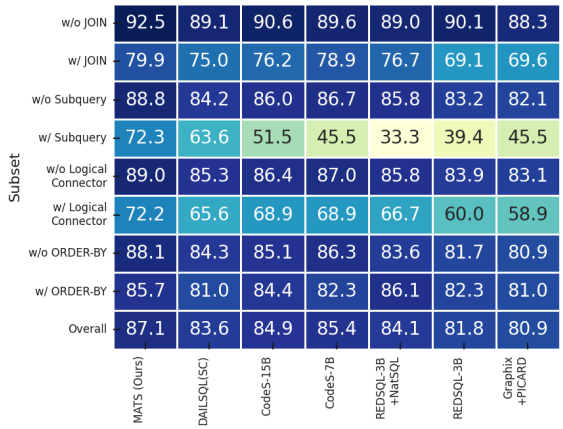


图 11：Spider 上不同 SQL 特征的执行准确率对比。



图 12：BIRD 上不同 SQL 特征的执行准确率对比。

基于人类反馈的强化学习。基于人类反馈的强化学习（RLHF）是一种广泛采用的方法，通过将人类评估直接融入训练过程，使语言模型与人类偏好保持一致

。在 RLHF 中，奖励模型通过人工标注数据进行训练，根据输出与用户意图的一致程度为其评分。随后，强化学习算法（例如近端策略优化（PPO））会利用这一奖励信号来优化模型的行为。RLHF 在推动语言模型生成更符合人类期望和偏好的回复方面发挥了重要作用。例如，这一方法已成功应用于 InstructGPT 等指令遵循模型，显著提升了它们在执行各种任务时遵循用户指南的能力 [29]。这项技术弥合了通用预训练与特定任务效用之间的差距，巩固了 RLHF 作为现代人工智能系统关键工具的地位。

受这一范式启发，我们提出了一种利用执行反馈（RLEF）的新型强化学习机制，以进一步改进 MATS 智能体。通过利用查询执行的响应，我们无需像基于人类反馈的强化学习（RLHF）那样投入人力提供标签。

无奖励模型对齐。为克服基于人类反馈的强化学习（RLHF）所伴随的复杂性和不稳定性，尤其是近端策略优化（PPO）[42], 带来的挑战，人们提出了替代性的对齐方法。直接策略优化（DPO）将奖励建模直接整合到偏好学习阶段，从而简化对齐流程，无需多阶段训练 [37]。ORPO [12] 在 DPO 的基础上，将基于优势比的惩罚项整合到监督微调中。ORPO 将 SFT 损失与相对比率损失相结合，无需单独的奖励模型或对齐阶段，从而简化训练并提升效率和可扩展性。

本文提出将 ORPO 应用于 Text2SQL 任务，并仅对补全部分施加损失。这样既能保持框架轻量，又能提高智能体的准确率，并简化训练流程，无需单独的奖励模型。

多智能体系统。 近期，多智能体系统的发展表明，将复杂任务分解为更小、更专业的子任务，并由单个智能体分别处理，能够有效解决复杂任务 [32, 48]。这些系统利用大语言模型的能力，促进智能体之间高效协作与沟通，从而提高任务准确率和效率。多智能体框架已成功应用于多个领域，例如在软件工程领域，ChatDev [35] 等工具有助于协作编码和调试；在社会模拟领域，多智能体系统模型则用于模拟人类互动，以评估政策结果 [31]。此外，多智能体系统也已应用于推荐系统，智能体通过有效协调和任务分配，协作处理用户偏好、筛选内容并生成个性化推荐 [53]。

受这些进展的启发，我们的框架采用多智能体方法处理 Text2SQL 任务，并为模式筛选、查询生成、验证和错误纠正分别分配专门的智能体。这种协作结构显著提高了 SQL 生成准确率，尤其适用于小语言模型，同时通过强化学习技术提升了可扩展性。

7 结论

在这项工作中，我们介绍了 MATS，这是一种专为 Text2SQL 任务设计的新型多智能体框架。该框架针对小语言模型进行了优化。与传统的基于大语言模型的方法不同，MATS 将 SQL 生成高效地拆分为由不同智能体角色负责的专门任务，从而能够在资源受限的环境中有效运行。具体而言，智能体会筛除无关的模式元素并检索相关列值；生成多个 SQL 查询；利用数据库响应评估其输出；优化 SQL 查询；最终从候选项中选出最佳 SQL 查询。此外，为进一步提升性能，我们提出了执行反馈强化学习，通过利用执行结果作为反馈来指导智能体对齐，从而改进 SQL 生成。通过在 BIRD 和 Spider 数据集上进行广泛评估，我们证明 MATS 的执行准确率可与大型闭源大语言模型相媲美，同时能够部署在单个 24GB GPU 上。结果表明，MATS 在执行准确率和鲁棒性方面优于其他开源模型，缩小了采用专有外部模型的闭源解决方案与开源方案之间的差距。

尽管取得了这些进展，要进一步提高准确率仍面临挑战。未来的工作将探索如何加强智能体协作并改进选择策略，以最大限度地提高执行准确率。我们还可以改进数据库系统本身的响应，以提升执行反馈的质量，帮助智能体更快、更好地学习。此外，我们将整合更广泛的领域特定知识，进一步提升 SQL 生成在实际应用中的表现。

参考文献

[1] Arian Askari、Amin Abolghasemi、Gabriella Pasi、Wessel Kraaij 和 Suzan Ver- berne。2023。将 BM25 分数作为文本注入，可提升基于 BERT 的重排序器的表现。发表于*ECIR*。66–83。[2] Shuaichen Chang、Jun Wang、Mingwen Dong、Lin Pan、Henghui Zhu 等。2023。Dr.Spider：面向文本转SQL鲁棒性的诊断评测基准。发表于*ICLR*。[3] Tri Dao、Dan Fu、Stefano Ermon、Atri Rudra 和 Christopher Ré。2022。Flashattention：具备 I/O 感知能力的快速、内存高效精确注意力机制。*NeurIPS* 35 (2022)，16344–16359。[4] Xiang Deng、Ahmed Hassan Awadallah、Christopher Meek、Oleksandr Polozov、Huan Sun 和 Matthew Richardson。2021。面向文本转SQL的结构基础预训练。发表于*NAACL-HLT*。1337–1350。[5] Guanting Dong、Hongyi Yuan、Keming Lu、Chengpeng Li、Mingfeng Xue、Dayi- heng Liu、Wei Wang、Zheng Yuan、Chang Zhou 和 Jingren Zhou。2024。监督微调数据组成如何影响大语言模型的能力。发表于*ACL*。177–198。[6] Xuemei Dong、Chao Zhang、Yuhang Ge、Yuren Mao、Yunjun Gao、Jinshu Lin、Dongfang Lou 等。2023。C3：使用 ChatGPT 实现零样本文本转 SQL。*arXiv*预印本 *arXiv:2307.07306* (2023)。[7] Yujian Gan、Xinyun Chen、Qiuping Huang、Matthew Purver、John R Woodward、Jinxia Xie 和 Pengsheng Huang。2021。提升文本转SQL模型抵御同义词替换的鲁棒性。*arXiv*预印本 *arXiv:2106.01065* (2021)。[8] Yujian Gan、Xinyun Chen 和 Matthew Purver。2021。探索跨领域文本转SQL泛化中尚未充分研究的局限。发表于*EMNLP*。8926–8931。[9] Dawei Gao、Haibin Wang、Yaliang Li、Xiuyu Sun、Yichen Qian、Bolin Ding 和 Jingren Zhou。2023。大语言模型赋能的文本转SQL：一项基准评测。*arXiv*预印本 *arXiv:2308.15363* (2023)。[10] Zihui Gu、Ju Fan、Nan Tang、Songyue Zhang、Yuxin Zhang、Zui Chen、Lei Cao、Guoliang Li、Sam Madden 和 Xiaoyong Du。2023。交替使用预训练语言模型和大语言模型进行零样本 nl2sql 生成。*arXiv*预印本 *arXiv:2306.08891* (2023)。[11] Taicheng Guo、Xiuying Chen、Yaqi Wang、Ruidi Chang、Shichao Pei、Nitesh V. Chawla、Olaf Wiest 和 Xiangliang Zhang。2024。基于大语言模型的多智能体：进展与挑战综述。发表于*IJCAI*。8048–8057。[12] Jiwoo Hong、Noah Lee 和 James Thorne。2024。ORPO：无需参考模型的单体偏好优化。发表于*EMNLP*。11170–11189。

[13] Md. Ashraful Islam、Mohammed Eunus Ali 和 Md Rizwan Parvez。2024。Map-Coder：用于竞赛问题求解的多智能体代码生成。发表于 ACL。4912–4944。[14] Ananya Kumar、Aditi Raghunathan、Robbie Matthew Jones、Tengyu Ma 和 Percy Liang。2022。微调可能会扭曲预训练特征，并降低分布外表现。发表于 ICLR。[15] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph E. Gonzalez、Hao Zhang 和 Ion Stoica。2023。借助 PagedAttention 高效管理大语言模型服务的内存。发表于 SIGOPS。[16] Wuwei Lan、Zhiguo Wang、Anuj Chauhan、Henghui Zhu、Alexander Li、Jiang Guo、Sheng Zhang、Chung-Wei Hang、Joseph Lilien、Yiqun Hu 等。2023。Unite：用于文本转SQL评估的统一基准。arXiv预印本 arXiv:2305.16265 (2023)。[17] Boyan Li、Yuyu Luo、Chengliang Chai、Guoliang Li 和 Nan Tang。2024。自然语言转SQL的黎明：我们准备好了吗？PVLDB 17, 11 (2024)，3318–3331。[18] Haoyang Li、Jing Zhang、Cuiping Li 和 Hong Chen。2023。Resdsql：解耦文本转SQL的模式链接与骨架解析。发表于 AAAI，第37卷。13067–13075。[19] Haoyang Li、Jing Zhang、Hanbing Liu、Ju Fan、Xiaokang Zhang、Jun Zhu、Renjie Wei、Hongyan Pan、Cuiping Li 和 Hong Chen。2024。CodeS：迈向构建用于文本转SQL的开源语言模型。SIGMOD 2, 3 (2024)，1–28。[20] Jinyang Li、Binyuan Hui、Ge Qu、Jiaxi Yang、Binhua Li、Bowen Li、Bailin Wang、Bowen Qin、Ruiying Geng、Nan Huo 等。2023。大语言模型是否已能充当数据库接口？面向大规模、以数据库为基础的文本转SQL任务的综合基准。NeurIPS 36 (2023)，42330–42357。[21] Jinyang Li、Binyuan Hui、Ge Qu、Jiaxi Yang、Binhua Li、Bowen Li、Bailin Wang、Bowen Qin、Ruiying Geng、Nan Huo 等。2024。大语言模型是否已能充当数据库接口？面向大规模、以数据库为基础的文本转SQL任务的综合基准。NeurIPS 36 (2024)。[22] Raymond Li、Loubna Ben Allal、Yangtian Zi、Niklas Muennighoff 等。2023。Starcoder：愿源代码与你同在！arXiv预印本 arXiv:2305.06161 (2023)。[23] Aiwei Liu、Xuming Hu、Lijie Wen 和 Philip S Yu。2023。全面评估 ChatGPT 的零样本文本转SQL能力。arXiv预印本 arXiv:2303.13547 (2023)。[24] Yinhan Liu。2019。Roberta：一种鲁棒优化的 BERT 预训练方法。arXiv预印本 arXiv:1907.11692 364 (2019)。[25] Zhenyan Lu、Xiang Li、Dongqi Cai、Rongjie Yi、Fangming Liu、Xiwen Zhang、Nicholas D Lane 和 Mengwei Xu。2024。小语言模型：综述、测量与洞见。arXiv预印本 arXiv:2409.15790 (2024)。[26] Aman Madaan、Niket Tandon、Prakhar Gupta、Skyler Hallinan、Luyu Gao、Sarah Wiegreffe、Uri Alon、Nouha Dziri、Shrimai Prabhumoye、Yiming Yang 等。2024。Self-refine：借助自反馈进行迭代改进。NeurIPS 36 (2024)。[27] Rudra Murthy、Prince Kumar、Praveen Venkateswaran 和 Danish Contractor。2024。使用知识任务评估语言模型的指令遵循能力。arXiv预印本 arXiv:2410.12972 (2024)。[28] Humza Naveed、Asad Ullah Khan、Shi Qiu、Muhammad Saqib、Saeed Anwar、Muhammad Usman、Naveed Akhtar、Nick Barnes 和 Ajmal Mian。2025。大语言模型全面概述。ACM智能系统与技术汇刊 16, 5 (2025)，1–72。[29] Long Ouyang、Jeffrey Wu、Xu Jiang、Diogo Almeida、Carroll Wainwright 等。2022。训练语言模型，使其通过人类反馈遵循指令。NeurIPS 35 (2022)，27730–27744。[30] Simone Papicchio、Paolo Papotti 和 Luca Cagliero。2025。Qatch：在专有数据上自动评估 SQL核心任务。ACM智能系统与技术汇刊 16, 2 (2025)，1–26。[31] Joon Sung Park、Joseph O'Brien、Carrie Jun Cai、Meredith Ringel Morris、Percy Liang 和 Michael S Bernstein。2023。生成式智能体：人类行为的交互式模拟体。发表于 UIST。1–22。[32] Kexing Peng、Shihao Zhu 和 Tinghuai Ma。2025。STPE-MARL：时空多智能体种群进化强化学习。ACM智能系统与技术汇刊 (2025)。[33] Mohammadreza Pourreza 和 Davood Rafiei。2023。DIN-SQL：通过自我纠错实现文本转SQL的分解式上下文学习。arXiv: 2304 11015 [ ] 。cs.CL

[34] Mohammadreza Pourreza 和 Davood Rafiei。2023。. DIN-SQL：通过自我纠错实现文本转SQL的分解式上下文学习。NeurIPS 36 (2023)，36339–36348。

[35] Chen Qian、Wei Liu、Hongzhang Liu、Nuo Chen、Yufan Dang、Jiahao Li、Cheng Yang、Weize Chen、Yusheng Su、Xin Cong 等。2024。ChatDev：用于软件开发的通信代理。载于 ACL。15174–15186。[36] Yiwei Qin、Kaiqiang Song、Yebowen Hu、Wenlin Yao、Sangwoo Cho、Xiaoyang Wang、Xuansheng Wu、Fei Liu、Pengfei Liu 和 Dong Yu。2024。InFoBench：评估大语言模型的指令遵循能力。载于 ACL。13025–13048。[37] Rafael Rafailov、Archit Sharma、Eric Mitchell、Christopher D Manning、Stefano Ermon 和 Chelsea Finn。2023。直接偏好优化：你的语言模型其实是一个隐藏的奖励模型。NeurIPS 36 (2023)，53728–53741。[38] Colin Raffel、Noam Shazeer、Adam Roberts、Katherine Lee、Sharan Narang、Michael Matena、Yanqi Zhou、Wei Li 和 Peter J Liu。2020。探索使用统一的文本到文本转换器进行迁移学习的极限。JMLR 21, 140 (2020)，1–67。[39] Karlis Rokis 和 Marite Kirikova。2022。低代码/无代码软件开发的挑战：文献综述。载于 BIR。3–17。[40] 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)。[41] Wout Schellaert、Fernando Martínez-Plumed 和 José Hernández-Orallo。2025。分析语言模型性能的可预测性。ACM智能系统与技术汇刊 16, 2 (2025)，1–26。[42] John Schulman、Filip Wolski、Prafulla Dhariwal、Alec Radford 和 Oleg Klimov。2017。近端策略优化算法。arXiv 预印本 arXiv:1707.06347 (2017)。[43] 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转SQL的适配（扩展版）。arXiv 预印本 arXiv:2306.00739 (2023)。[44] Shayan Talaei、Mohammadreza Pourreza、Yu-Chen Chang、Azalia Mirhoseini 和 Amin Saberi。2024。CHESS：用于高效 SQL 合成的上下文利用框架。arXiv 预印本 arXiv:2405.16755 (2024)。[45] Hugo Touvron、Thibaut Lavril、Gautier Izacard、Xavier Martinet、Marie-Anne Lachaux、Timothée Lacroix、Baptiste Rozière、Naman Goyal、Eric Hambro、Faisal Azhar 等。2023。Llama：开放且高效的基础语言模型。arXiv 预印本 arXiv:2302.13971 (2023)。[46] Bing Wang、Changyu Ren、Jian Yang、Xinnian Liang、Jiaqi Bai、Qian-Wen Zhang、Zhao Yan 和 Zhoujun Li。2023。MAC-SQL：面向文本转SQL的多智能体协作。arXiv 预印本 arXiv:2312.11242 (2023)。[47] Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Fei Xia、Ed Chi、Quoc V Le、Denny Zhou 等。2022。思维链提示能够引导大语言模型进行推理。NeurIPS 35 (2022)，24824–24837。[48] Zhaohui Wei、Lizi Liao、Xinguang Xiang 和 Xiaoyu Du。2025。利用众包知识丰富面向任务的对话代理的回复。ACM智能系统与技术汇刊 16, 2 (2025)，1–24。[49] 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)。[50] 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。载于 NAACL-HLT。588–594。[51] Tao Yu、Michihiro Yasunaga、Kai Yang、Rui Zhang、Dongxu Wang、Zifan Li 和 Dragomir Radev。2018。SyntaxSQLNet：用于复杂跨领域文本到SQL任务的语法树网络。载于 EMNLP。1653–1663。[52] Tao Yu、Rui Zhang、Kai Yang、Michihiro Yasunaga、Dongxu Wang、Zifan Li、James Ma、Irene Li、Qingning Yao、Shanelle Roman、Zilin Zhang 和 Dragomir Radev。2018。Spider：面向复杂跨领域语义解析和文本转SQL任务的大规模人工标注数据集。载于 EMNLP。3911–3921。[53] Junjie Zhang、Yupeng Hou、Ruobing Xie、Wenqi Sun、Julian McAuley、Wayne Xin Zhao、Leyu Lin 和 Ji-Rong Wen。2024。AgentCF：自主语言代理在推荐系统中的协作学习。载于 WWW。3679–3689。[54] Victor Zhong、Caiming Xiong 和 Richard Socher。2017。Seq2SQL：利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)。[55] Jeffrey Zhou、Tianjian Lu、Swaroop Mishra、Siddhartha Brahma、Sujoy Basu、Yi Luan、Denny Zhou 和 Le Hou。2023。大语言模型的指令遵循评测。arXiv 预印本 arXiv:2311.07911 (2023)。