用于 SQL 生成的反思性推理

IsabelleMohr[1,2], **Joao PedroGandarela**[1], **JohnDujany**[2], **AndreFreitas**[1,]
Idiap 研究院, [2]默克 KGaA, **通信作者：** isabelle.mohr@idiap.ch

arXiv:2601.06678v1 [cs.DB] 10 Jan 2026

摘要

即使使用现代大语言模型，在复杂的真实世界数据库上进行稳健的文本到SQL转换仍然十分脆弱：迭代式优化往往会引入句法与语义漂移，修正通常无法迁移到其他查询，而简单地使用大上下文窗口的扩展性也较差。我们提出了一种围绕反思式改进构建的受控文本到SQL框架。系统不再反复重写当前的 SQL 实例，而是将生成过程分解为类型化阶段，并将反馈作为对阶段级生成机制的持久更新。反思-细化循环将违规定位到相应阶段，最大限度地保留先前已验证的约束，并支持在查询集上实现单调改进。该方法无需标准 SQL，而是结合基于解释器的检查与基于大语言模型的语义覆盖率验证，将其作为认识论的判断器。在 Spider 和 BIRD 上进行的实验表明，与强大的提示基线相比，该方法取得了持续的提升，能够在较小的细化预算内稳健收敛，并且在前沿模型和开放权重模型系列上都提高了执行准确率。

1 引言

尽管大语言模型（LLM）显著推动了面向数据库的自然语言接口的发展，但在复杂或真实世界的数据库上实现稳健的文本到SQL生成仍未得到解决（Yu 等人，2019年；Wang 等人，2021）。

近期研究越来越多地从单次生成的语义解析转向由大语言模型驱动的多阶段文本到 SQL 流水线，这类流水线会将任务分解，并纳入迭代式纠错。基于分解的提示和自我纠正通过将模式解释、谓词落地和 SQL 构造分开处理来提升鲁棒性（Pourreza 和 Rafiei，2023a年），而执行引导优化利用运行时信号诊断并增强可靠性修复错误（Mao 等人，2024年）。与此同时，将推理与反馈交织进行的智能体推理模式，使迭代式优化成为一种广受采用的通用策略（Yao 等人，2023年）。然而，这些方法通常通过重写特定的 *SQL* 实例来进行优化，往往采用部分或完全重新生成的方式，而非改进底层的阶段提示词或机制，因此纠正结果可能较为脆弱，且难以迁移到其他查询（Wang 等人，2023年）。

现象：为什么文本到SQL本质上是一个难题?

（挑战 1）迭代式优化过程中的语法／语义漂移。近期基于迭代式优化的方法（Wang 等人，2023年；Yao 等人，2023年）仍难以在多轮迭代中实现生成控制，新的迭代周期往往会引入新的不一致。

（挑战 2）缺乏单调泛化机制。大多数方法侧重于针对每个查询从头构建映射／解决方案，却没有形成可持久泛化并能在数据库内的下游查询中重复利用的机制。

（挑战 3）语义映射与大上下文窗口。文本到 SQL 的核心挑战在于如何解决自然语言查询指代与异构模式之间的语义映射，整合多个表，解读可能涉及大型数据库的属性/值域表达式。对大上下文窗口不加节制地使用，会增加性能下降（幻觉）的概率，对于大型数据库而言最终也难以实现。

该机制：基于局部化反馈进行反思式自我纠正推理。为了应对大语言模型引导的文本到SQL 生成所面临的挑战，我们提出将三种干预方式紧密结合：

反馈定位。 针对 挑战 *1*，我们将文本到SQL过程分解为一系列定义明确的查询生成阶段（图 1(a)），这些阶段可以按模块化方式进行干预（而非整体重新生成查询）。错误由 批评分析器 机制识别、定位并诊断。

反思式改进。 针对 挑战 *2*，我们提出一种反思式改进机制，利用批评分析器生成的细粒度诊断上下文，对每个查询生成阶段进行持续调整。作为一种集成机制，批评器–改进器循环会定位违规之处，反思未能满足的约束，并有选择地重新调用唯一需要负责的阶段，从而实现有针对性的修正，同时不破坏此前已验证的语义，进而使查询生成过程得以反思式单调改进。这使我们能够建立一种机制，对更宏大的语义解析问题进行抽象。

数据库上下文代理。此外，为应对挑战 *3*，我们提供了一种数据库表示机制，将数据库模式与值/内容分离，以尽量减少上下文窗口的使用（图1 (c)），并允许对模式级和取值级上下文进行可控解读。

由这些组件定义的机制（图1）支持一个自我纠正反思循环，能够对错误进行抽象，并在查询合成/生成过程中实现覆盖率的单调提升。所提出的方法无需依赖监督式黄金标准进行训练，而是使用代理评估函数（针对语法正确性、语义保留以及模块化且局部化的错误分析）作为自我纠正的反馈代理。在最先进的关系型基准测试上进行的实验表明，该方法在不同模型系列中均取得稳定提升、稳健收敛，并能进行细粒度的错误归因。

本工作的贡献如下：

- 我们介绍了**反思性细化**机制，它可以抽象处理错误并细化文本到 SQL 生成过程（而不是局部查询）。
- 我们描述并形式化了所提出的方法。
- 我们使用最先进的基准测试，对该方法进行了全面的实证分析。

本文其余部分的结构如下。第 2节将文本到 SQL 生成形式化为约束满足问题，并引入单调细化和语义覆盖的概念。第 3 节和 4 节介绍所提出的受控多阶段推理框架，包括数据库模式解释、谓词落地、查询构建和反思式改进。第 5 节介绍实验设置和基准测试；第 6节则分析收敛行为、反馈粒度和各组件的贡献。最后，第8 节讨论局限性和未来方向，并作出总结。所有补充材料均可在附录中找到。

为支持可复现性，我们公开了本研究所用的完整代码库、提示词、实验配置和评估脚本，并附有复现所有报告结果的说明。[1]

2 预备知识

问题定义。 设 $\mathcal{DB}$ 为一个关系数据库。给定一个自然语言查询 q，文本到 SQL 任务旨在生成一个可执行 SQL 查询 Q ，使其在 $\mathcal{DB}$ 上的执行语义与 q所表达的意图相符。我们将其建模为一种映射

$$\mu : \mathcal{Q} \times \mathcal{DB} \rightarrow \mathcal{L}_{\mathrm{SQL}}(\mathcal{DB}), \qquad (q, \mathcal{DB}) \mapsto Q, \tag{1}$$

其中， $\mathcal{Q}$ 是自然语言查询空间， $\mathcal{L}_{\mathrm{SQL}}(\mathcal{DB})$ 是 $\mathcal{DB}$上的可执行 SQL 查询集合。正确性是语义上的（指称性的）：在执行语义 $[\![\cdot]\!]_{\mathcal{DB}}$下，

$$[\![\mu(q, \mathcal{DB})]\!]_{\mathcal{DB}} \approx \mathrm{Ans}(Q, \mathcal{DB}), \tag{2}$$

其中，Ans(Q， $\mathcal{DB}$) 表示预期答案，而 $\approx$ 则抽象掉表层形式上的差异。

[1]为匿名起见，省略链接。

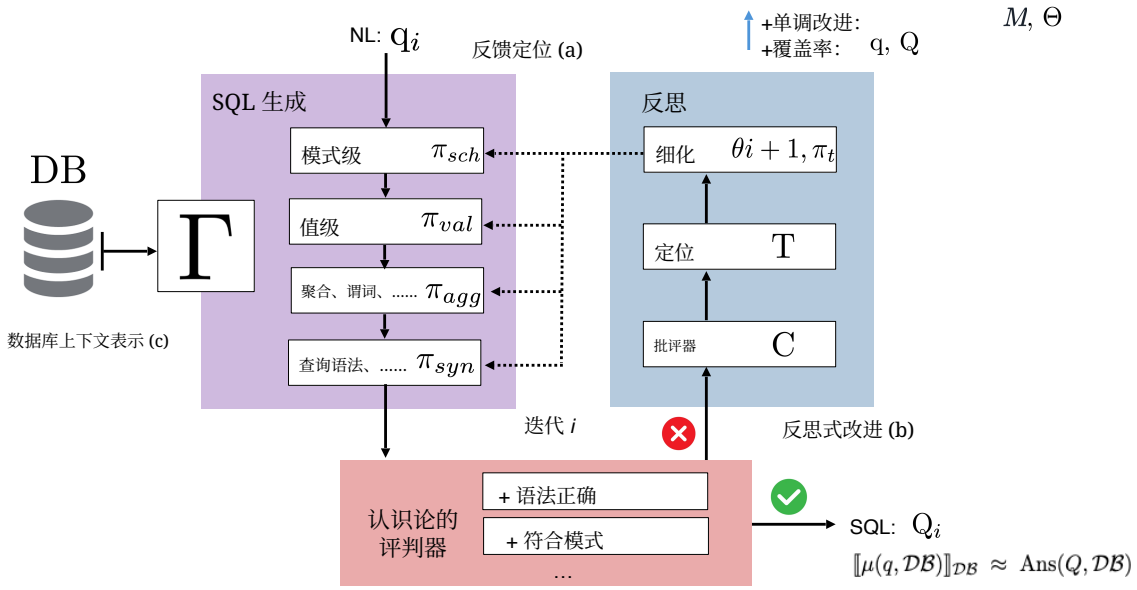


图 1：反思式文本到 SQL 方法的高层概览，展示了分阶段生成、批评器反馈、语义检查和迭代式优化。

组合式生成。 我们不将 μ 视为单一的函数合成器来归纳，而是将其表示为由类型化生成函数 $\{\pi_t\}_{t=1}^{T}$组成的组合。整体映射如下：

$$\mu \triangleq \pi_T \circ \pi_{T-1} \circ \cdots \circ \pi_1. \tag{3}$$

局部化生成函数。 我们假设整体映射 μ 可分解为若干类型化生成函数 $\pi^{(t)}$的组合，每个函数面向文本到 SQL 问题的一个互不相交的抽象层：$\pi^{(\text{SCH})}$ 用于模式级映射（指代表、属性、连接结构），$\pi^{(\text{VAL})}$ 用于值/实例级落地（属性域和规范值），$\pi^{(\text{AGG})}$ 用于聚合和投影约束，$\pi^{(\text{PRED})}$ 用于谓词约束（筛选条件、字面量、运算符），以及 $\pi^{(\text{SQL})}$ 用于在目标语言 $\mathcal{L}$（SQL）的语法约束下进行最终的声明式实现 $q \mapsto Q$。

3 反思式改进

概述。 图 1展示了整体架构。给定一个自然语言问题 q 和一个数据库 $\mathcal{DB}$，系统首先执行一组初始查询生成函数，随后进行验证步骤。然后，系统利用评估代理确定查询的语法和语义正确性。如果检测到违规，反思循环将通过反思和抽象来诊断并定位错误，进而引导一个仅针对特定查询生成函数的细化过程，在最大限度保留先前已验证约束的同时进行细化。当生成的查询同时满足结构检查和语义检查，或达到固定的细化预算时，该过程终止。细化循环的完整过程规范见附录 1。

反思式改进。给定一个查询集$\{(q_i,\mathcal{DB})\}_i$ 以及生成器 π_θ^t 的一组当前约束$\Theta = \{\theta_t\}_{t\in\mathcal{T}}$（因此有 μ_Θ）。请注意，θ_t 是与类型化生成函数 π^t相关联的提示组件。每个细化周期按如下方式进行：

1. 生成：针对每个 i，生成 $Q_i \leftarrow\mu_\Theta(q_i,\mathcal{DB}_i)$。2. 评估：获取自然语言诊断信息 $r_i \leftarrow \text{eval}(Q_i,\mathcal{DB}_i,q_i)$，其中 eval 包含语法／执行组件和语义（大语言模型）意图保留组件，即$[[\mu(q,\mathcal{DB})]]_{\mathcal{DB}} \approx \text{Ans}(Q,\mathcal{DB})$。3. 批评：识别查询不一致：$\{(t,\, c,\, Q_i')\} \leftarrow f_{critic}(r_i,\Gamma_{\mathcal{DB}},Q_i)$ 4. 定位：识别目标组件类型：$\pi_t \leftarrow \ell(r_i) \in \mathcal{T}$. 5. 反思式细化：仅更新 π_t 中受到影响的参数：$\theta_{t,i+1} \leftarrow \text{Reflect}(\theta_t, i; r_i)$,

6.**最大化反向保留：**同时确保所有 θ_t, i 在语义上均得到保留。

与标准细化的本质区别在于，修改针对的是机制参数 θ_t （例如 $\pi^{(t)}$ 的提示词），而不是特定的查询实例 Q_i。

反思。构建查询假设 Q_i 后，系统通过批评器-细化器循环评估候选查询。批评器是一种基于上下文学习的大语言模型解释函数 f_{critic} ，它依据一组标准（$\mathcal{C}$）对照 $\mathcal{DB}$ 上下文代理 Γ（参见附录 D.3），对查询进行系统检查。这些标准 $\mathcal{C}$ 由一组关键评估函数定义，并针对每种 $\mathcal{T}$ 类型分别设定：(i) 模式级违规，即无效的表和属性引用、连接；(ii) 值级违规，即内容级引用中的标识符和格式不一致；以及 (iii) 聚合与谓词语义；(iv) 查询语法违规。批评器返回一个已定位的违规集合 $\{(t, c, Q_i')\}$，$t \in \mathcal{T}$，$c \in \mathcal{C}_t$，$Q_i' \subseteq Q_i$，其中 t 表示被违反的抽象类型，c表示具体标准，Q' 表示涉及的查询片段（例如跨度/子句）。相应的批评器实现详见附录 D.5。

抽象与细化。 这种选择性重启机制不同于通用重试策略：它利用上一步的局部诊断输入，对具有查询集 θ_i 中π_t覆盖率的参数集合 q_i，Q_i并在查询生成组件（θ_{i+1}）内更新这些参数（π_t），同时最大限度地保留正确推理步骤。相应的细化函数实现见附录 1。

通过认知判断器提供无监督反馈。由于不假设存在标准 SQL，进展仅通过评估来定义，将其作为语法正确性和语义保持的代理函数。本研究将近期提出的形式认识论评判器概念（ ）应用于查询合成领域。形式认识论评判器最初在数学自动形式化领域提出，旨在提供一组评估函数，用作复杂任务的细粒度评估代理。认识论判断器可以基于大语言模型（L），也可以基于解释器（I）。在本研究中，语法正确性（I）和语义保持（在 q_i， Q_i 之间）的评估函数通过（L）实现。

4 数据库上下文解读

关系数据库 $\mathcal{DB}$ 通常过于庞大且异构，无法“原样”提供给大语言模型：完整模式可能很长，而实例级内容实际上没有边界。因此，为了实现受控的文本到 SQL 生成，我们用紧凑且可复用的上下文代理 Γ 替代 $\mathcal{DB}$，该代理（i）保留决定生成内容是否合法的模式约束，并且（ii）仅提供经过精心预算的实例级证据样本，以便为属性的值域提供落地依据。具体而言，我们将缓存上下文 Γ 构建为离线预处理步骤；Γ 是下游生成器 π_t 访问$\mathcal{DB}$的接口。

上下文代理。 给定一个数据库 $\mathcal{DB}$，我们定义一种上下文构造映射

$$\Gamma \;=\; \Gamma(\mathcal{DB}), \tag{4}$$

其中，Γ 捆绑了模式级和值级的解释/采样函数。直观而言，Γ 是一种紧凑表示，支持受控地访问 (a) 模式结构和 (b) 有限的值证据，而无需对整个数据库进行端到端编码。

模式级接口。 Γ 提供用于规范化和概括模式元素并约束可行连接的函数：

$$\Gamma_{\text{SCH}} \;=\; \big(f_{\Delta},\, f_{\Sigma},\, f_{\mathcal{J}}\big), \tag{5}$$

其中，f_{Δ} 生成规范化的表/列描述符（规范名称、类型、键可能性），f_{Σ} 生成简洁的自然语言表摘要，以支持可扩展的表选择；$f_{\mathcal{J}}$根据已声明的外键生成连接候选项（可选择用样本加以佐证），以限制可行的连接路径。

值级接口（预算证据）。当样本可用时，Γ 还会提供一个基数感知值落地接口：

$$\Gamma_{\text{VAL}} \;=\; f_{\mathcal{V}}, \tag{6}$$

其中，$f_{\mathcal{V}}$ 会展示低基数属性（如状态/类别字段）的代表性值，同时省略高基数属性（如名称）的枚举，以控制上下文长度。这样便形成了用于字面量落地的受控词汇表，无需在提示中扩充大量值列表。

用途。 下游阶段将 Γ 作为唯一的面向数据库的上下文：（a）借助归一化描述符和摘要来指导表/属性的选择；（b）连接仅限于 $f_{\mathcal{J}}$许可的候选项；（c）在固定预算下，通过 $f_{\mathcal{V}}$ 改进字面量/约束落地。从这个意义上说， Γ（$\mathcal{DB}$）是 $\mathcal{DB}$的实用代理，既能提供准确生成所需的充分证据，又能确保在查询集中的可控性和可复用性。

5 实验

我们在两个多领域文本到SQL基准——Spider 1.0 和 BIRD 上评估 REFLECT-SQL，以检验 (i) 端到端执行准确率，(ii) 迭代式优化过程中的收敛性，以及 (iii) 反馈和语义覆盖检查的作用。

5.1 设置

基准测试。 我们使用 Spider 1.0 和 BIRD 官方开发集划分。Spider 1.0 开发集包含 1,034 个问题，涵盖 20 个数据库，这些数据库选自基准测试的 200 个跨领域模式集合，着重考察模型在异构结构上的组合泛化能力。BIRD 共包含 12,751 个数据对，涉及 95 个数据库，并加入了更具挑战性的 SQL 模式，同时评估查询有效性和效率，为复杂关系推理提供互补的压力测试。

基线。 我们将其与基于 GPT-4 的强提示系统进行比较：DIN-SQL (Pourreza 和 Rafiei，2023b)、DAIL-SQL (Gao 等人，2023年)、MAC-SQL (Wang 等人，2025a) 和 MCS-SQL (Lee 等人，2024年)，以及 GPT-4 零样本基线 (OpenAI 等人，2024年)。

模型与解码。我们使用一组具有代表性的基础模型来实例化 REFLECT- SQL，涵盖专有模型和开放权重模型。GPT-4 和 GPT-5 级别的模型作为强有力的上界参考，用于衡量最先进的通用大语言模型在受控提示下的性能，并将我们的方法与以往的最先进方法进行比较。同时，我们评估了参数规模约为 30B 的有竞争力的开放权重模型（例如 Qwen3-30B 和 DeepSeek- R1-32B）。这类模型在能力与易获取性之间实现了良好平衡，并常用于近期的文本到 SQL 系统。

方法	BIRD		Spider
	VES	EX	EX
GPT-4（零样本）	49.8	46.4	74.6
DIN-SQL	58.8	50.7	82.8
DAIL-SQL	56.1	54.8	84.4
MAC-SQL	58.8	57.7	86.8
MCS-SQL	64.8	63.4	89.5
REFLECT-SQL（基础版）	58.5	54.6	81.7
REFLECT-SQL（完整版）	66.5	65.2	93.8
— GPT-5	**68.9**	**67.8**	**95.4**
— Qwen3-30B	61.2	59.9	92.1
— DeepSeek-R1-32B	63.7	62.1	94.3

表 1：**完整的 REFLECT-SQL 持续优于先前的提示方法和智能体基线，证明反思式改进的益处不止于分阶段分解。Spider 1.0 和 BIRD 上的基线与最先进方法对比，报告有效效率分数（VES）和执行准确率（EX）。除非针对 REFLECT-SQL 变体另有说明，所有方法均使用 GPT-4 作为基础模型。**

为检验所提出的机制是否依赖模型规模，还是在容量受限时仍然有效，我们还使用一个规模小得多的开放权重模型（LLaMA 3.1 8B）进行了有针对性的消融实验。通过这一设置，我们可以验证，受控分解、语义规划和选择性精炼带来的优势不仅适用于前沿规模的模型，而且在基础模型的推理能力有限时依然有效。

除非另有说明，所有阶段均采用确定性解码（温度 ≈ 0），并为每个阶段设定固定的词元预算，以降低方差并确保模型间的可比性。

细化预算。每个示例最多允许 T 步细化；当结构检查和语义覆盖检查通过时提前停止。优化器会根据批评器/分析器信号，仅有选择地重新运行受到影响的阶段（模式解释、谓词落地或查询构造）。

指标。我们报告两个基准测试上的执行准确率（EX），以及 BIRD 上的有效效率分数（VES）。

5.2 主要结果

在所有基准测试中，REFLECT-SQL 的表现均优于所有提示基线和先前的智能体系统。包含反馈细化迭代次数在内的完整方法始终优于分阶段基线，这表明反思式细化和定向重启具有显著价值，

其价值超越了单纯分解所能带来的提升。这些优势在不同模型系列中均能保持，说明该方法并不依赖于特定的基础模型。

6 分析

6.1 细化过程中的收敛

在所有基础模型和数据集中，执行准确率（EX）均随着细化预算的增加而持续提升。图2绘制了EX 随细化迭代次数变化的曲线。在 Spider 1.0 上，EX 的平均 $\sim$ 12 个百分点提升发生在迭代次数 $t = 0$ 至 $t = 3$ 之间（从$\sim$ 85 至 $\sim$ 95 GPT5 的 EX，以及从 $\sim$ 82 至$\sim$ 94 GPT4o 的 EX），之后性能趋于饱和，并且对于较弱的模型可能出现下降（见 Llama3.1-8B）。Qwen3-30B 和 DeepSeek-R1-32B等中等规模模型表现出更大的相对增益（例如，DeepSeek-R1-32B的 EX 从 72.4 提升至 89.5，经过 $t = 3$），这表明，当单次生成性能仍远未达到饱和时，智能体细化循环尤其有益。Qwen3-8B 和`llama3.1-8B` 等较小模型在最初三次迭代中也能获得显著收益（例如，Qwen3-8B 的 EX 从 59.5 提升至 78.4，经过 $t = 3$），但在 $t = 3$ 之后，其曲线相比更大规模的模型表现出更明显的波动。

超过 $t = 3$ 后，在 Spider 1.0 上继续迭代的收益递减，甚至可能损害性能。对于最强的模型（GPT5、GPT4o、DeepSeek-R1-32B），在 $t = 3$ 之后，执行准确率（EX）保持稳定或略有下降，这表明模型已经收敛，之后的编辑大多只会造成无关紧要的扰动。相比之下，较弱的模型则表现出明显的过度细化迹象：Qwen3-8B 和 llama3.1-8B 在 $t = 3$ 到 $t = 6$ 之间的执行准确率下降了 5–8 个百分点。从定性分析来看，我们发现有些情况下，较早的一次迭代引入了幻觉表、连接或谓词；随后的迭代便围绕这一错误假设进行反思和细化，而不是将其纠正，导致系统“锁定”在错误的执行计划上。

在 BIRD（右图）上的表现也呈现出相同趋势，但绝对得分更低，性能下降也稍晚。所有模型的起始表现都显著低于其在 Spider 1.0 上的对应表现（例如，GPT5 的执行准确率为 56.2，GPT4o 为 54.6），这反映了 BIRD 更高的模式复杂度和 SQL 难度。不过，最初三到四个细化步骤仍带来了显著提升：GPT5 到 $t = 3$ 时达到 67.8 的执行准确率，并且此后保持稳定，而

反馈风格	BIRD (EX)	Spider (EX)	平均迭代次数 (Spider)
粗略	61.0	90.7	2.3
细粒度	**65.2**	**93.8**	**3.1**
仅认知	64.7	92.5	2.6

表 2：**与粗略反馈或仅认知反馈相比，局部化的分类反馈能够实现更精确的修正和更高的执行准确率，但需要略多的细化迭代**。 在 Spider 1.0 和 BIRD 上，针对固定基准模型（GPT4o）比较粗略和细粒度分类反馈。

GPT4o 到 $t = 4$时升至 65.2 的执行准确率。较大的开源模型（Qwen3-30B、DeepSeek-R1-32B）呈现出相似的轨迹，在约 $t = 3$–4 时达到峰值，之后略有下降。较小的模型（Qwen3-8B、llama3.1-8B）在 $t = 3$之前提升了 10–15 个执行准确率积分，但在 $t \geq 5$时性能开始明显下降，这同样符合这样的情况：围绕脆弱或幻觉式中间信念进行过度细化。

总体而言，这些曲线表明：（i）较小的细化预算（Spider 1.0 上为 $t \approx 3$，BIRD 上为 $t \approx 3$–4）即可获得大部分可实现的收益；（ii）对于强模型，将迭代预算增加到此范围以上几乎没有益处，且可能损害较弱模型的表现；（iii）智能体循环会系统性地缩小小模型与大模型之间的差距，但无法消除差距。实际应用中，这意味着应采用适中且针对模型和基准测试进行调整的迭代上限，而不是让细化循环不间断地运行。

6.2 反馈粒度

高度定位的分类反馈始终能提升执行准确率，但代价是细化迭代次数略有增加。 表 2比较了固定基础模型（GPT4o）在 BIRD 和 Spider 1.0 上的三种反馈方式。粗粒度反馈仅标记整个候选结果正确或错误，对错误的定位极少，但已取得出色表现（执行准确率分别为 61.0 和 90.7），平均细化 2.3 步。细粒度反馈包含类别特定信号和跨度级信号（例如指出错误的连接、谓词或缺失的约束），与粗粒度反馈相比，在两个基准测试上都能稳定提升表现，但需要略多的细化步骤（3.1 次迭代）。这是意料之中的：信息量更大的反馈能够让

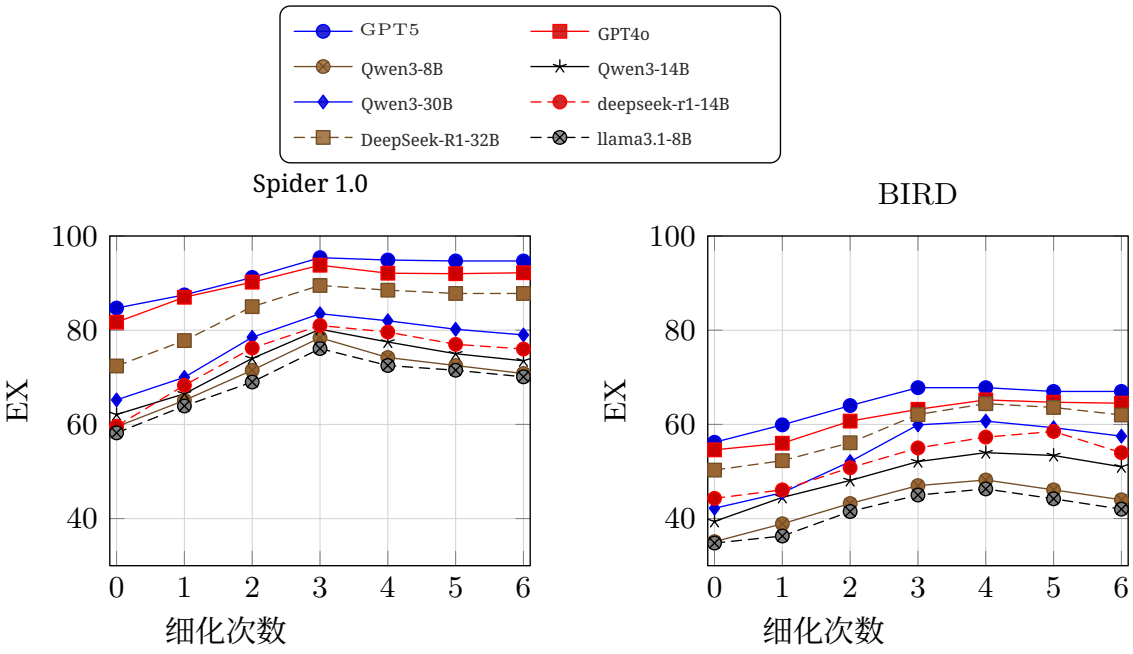


图 2：**在所有模型中，早期细化带来了大部分收益，而过度迭代会导致收益递减或过度精炼，因此需要采用选择性且有界的细化**。每个细化步骤都会报告执行准确率 t，适用于 Spider1.0（开发集）和 BIRD（开发集）上的一系列基础模型。

模型在每一步进行更有针对性的修正，但也会促使其持续细化，直到获得质量更高的解决方案。

仅认知反馈侧重于纠正模型对模式解读和问题的底层假设及高层概括（例如，不匹配的实体类型或误解的关系），但不提供完全细粒度的结构提示，处于两种极端之间：平均仅进行 2.6 次细化迭代，却分别达到 64.7 和 92.5 的执行准确率。

总体而言，细粒度反馈能够更可靠地纠正问题，因为它能准确指出需要修改哪些语义约束，而不是触发全局重新生成。

6.3 各组件的贡献

性能提升源于分阶段分解、局部优化和意图层语义验证的结合，而非任何单个组件单独发挥作用。为定位性能提升的来源，我们对主要组件进行了消融实验（表3）。移除批评器–改进器循环后，VES 和执行准确率均明显下降，表明基于执行的反馈和诊断对于纠正生成的 SQL 查询至关重要。禁用语义检查器会导致更大的性能下降（与完整方法），凸显了除语法和执行之外，意图级验证的重要性。最后，将系统简化为不含分阶段约束的单次生成器会导致最大的性能损失，这证实了分解本身是实现稳健细化的必要（但不充分）条件。

配置	VES	EX	ΔVES	ΔEX
完整方法	**66.5**	**65.2**		
无批评器（仅语法检查）	62.4	61.6	−4.1	−3.6
无语义检查器	63.1	58.2	−3.4	−7.0
单次生成（无阶段）	58.5	54.6	−8.0	−10.6

表3：**移除分阶段分解、批评器–改进器循环或语义检查器后，准确率会下降**，这表明稳健的性能提升源于它们之间的交互，而非任何单个组件单独发挥作用。BIRD 上的组件消融实验。每一行移除完整流水线中的一个或多个组件；Δ 列显示相对于完整系统的性能变化。

综合来看，可靠的改进源于约束生成阶段与针对性反思式改进之间的交互，而非单靠分解或验证。

6.4 定性案例与基准局限性

许多看似是文本到SQL“错误”的情况，实际上源于评估规范不充分，而非真正的语义失败，这凸显了进行意图级验证的必要性。精确匹配评估将真正的语义失败与以下情况混为一谈：(i) 有效但不同的 SQL 表达形式；(ii) Spider 1.0 中存疑的标准答案标注。表4 展示了三种反复出现的模式。第一，**模式/连接歧义**：多个连接路径可能产生等价的指称结果，但基准测试往往只提供一个规范查询。第二，**边界和字面量错误**：真正的失败往往局限于运算符选择或字面量落地（例如，$>$ 与 $\geq$），这正是我们的分阶段设计能够归因于谓词落地的错误类型。第三，**过度/不足指定**：模型输出可能包含标准 SQL 中没有的合理约束（或省略标准 SQL 隐含假设的约束），这揭示了基准测试的最简原则与实际意图解读之间的不匹配。这些观察促使我们采用语义覆盖检查，以评估必要约束是否存在，而不受表层形式影响。

7 相关工作

模式链接与落地。 将提及内容与模式/值证据对齐仍是一个核心瓶颈。LinkAlign 将模式链接扩展到大规模多数据库环境（Wang 等人，2025年b），而上下文模式链接图则提升了 ICL 示例选择与落地效果（Lee 等人，2025年）。内容感知提示（例如，基于值落地的问题改写）可在生成前进一步消除歧义（Mao 等人，2024年）。

分解、规划与结构化上下文学习。 许多系统通过逐步分解与自我纠正来提升可靠性。DIN-SQL 通过迭代纠正实现分解式 ICL（Pourreza 和 Rafiei，2023年a）；PTD-SQL 按查询类型划分，以进行有针对性的“深入分析”（Luo 等人，2024年）；ROUTE 通过稳健的多任务调优与协作，增强跨子任务泛化的稳定性（Qin 等人，2025年）。

交互与多轮文本到SQL。对话式文本到SQL需要上下文延续和增量编辑；CoE-SQL 将其建模为基于先前查询构成的 SQL 编辑链（Zhanget al.，2024年）。

执行引导验证与智能体细化循环。执行反馈对于细化和监督日益重要。DART- SQL 通过执行引导细化修复候选项（Mao 等人，2024年），而 ExeSQL 利用执行验证，在不同 SQL 方言中引导生成自学习数据（Zhang 等人，2025年）。通过偏好优化的选择进行多候选项推理，可进一步提高最终执行准确率（Pourreza 等人，2025年）。

面向开放模型的数据合成与微调策略。缩小开源与闭源差距有赖于合成监督数据和针对性调优。DTS-SQL 采用分阶段分解来支持小型大语言模型（PourrezaandRafiei，2024年），而弱/强大语言模型合成流水线则为指令调优提供了可扩展的训练数据（Yang etal.，2024年）。

本文的定位。在这些研究方向中，先前系统主要通过加强落地（模式/值链接）、将生成分解为多个阶段，再通过修复或重新生成当前 *SQL* 实例来应用反馈，或通过合成/调优引入额外监督，从而改进文本到SQL。相比之下，我们的核心贡献是将细化视为对阶段级生成机制 （类型化生成器的提示/参数更新）的调整，这种调整由局部诊断和抽象驱动，并借助认知代理（无需假设存在标准 SQL或额外训练），使改进能够迁移到数据库中的一组查询，而非局限于脆弱的逐查询编辑。

8 结论

本文将文本到 SQL 的迭代改进重新界定为一种反思式细化机制，而非实例级 SQL 重写。针对当前由大语言模型驱动的流水线的三种核心失败模式——细化过程中的偏移、缺乏可迁移的泛化能力，以及模式/值落地的可扩展性较差——我们引入了受控的分阶段生成流程，并将其与能够定位违规的反思—改进循环相结合。反思式改进的核心理念是：利用已定位的诊断上下文更新各阶段生成器的参数（例如提示约束），同时最大限度地反向保持已验证的约束，从而在数据库内的查询集上实现覆盖率的单调提升。作为补充，我们提出了一种紧凑且可复用的数据库上下文代理，将模式结构与采样值/领域

证据解耦，从而减少对上下文窗口的依赖，并改善受控落地。

在本文所提出的设计范围内，最终系统形成了一个比单体式重试策略更易控制、也更易复用的自我纠正循环：细化针对负有责任的最小抽象层，并通过无监督的认知判断器评估进展，这些判断器结合了基于解释器的语法与执行检查，以及意图层级语义覆盖验证。这种组合机制为实现稳健的文本到SQL行为提供了切实可行的途径，无需假设存在黄金SQL监督，同时兼容各种基础模型。除文本到SQL之外，该机制还为结构化预测提供了一种通用范式：通过局部诊断以及对底层生成算子的保持约束的更新，推动迭代改进。

局限性

所提出的方法受固定细化预算的限制；即使预算耗尽后问题仍然存在，也会返回最终 SQL 候选项，因此一些棘手的情况可能无法解决。其有效性取决于批评器和分析器组件的质量：不准确或过于粗略的诊断信息可能会将失败归因于错误的抽象层，导致细化效率低下或遗漏修正。此外，当前的语义检查器会强制执行一组预定义的正确性约束（例如模式一致性、连接可行性、聚合语义），但可能无法检测到更隐蔽的语义不匹配；这些不匹配虽然仍可执行，却偏离了用户意图。尽管该框架有意围绕语义忠实性和可控性进行设计，但它并未明确推理查询效率或执行成本。将面向效率的认知判断器纳入其中（例如基于查询计划或执行统计信息）仍是未来工作的重要方向。最后，尽管本文展示的是该框架在自然语言到SQL任务中的应用，但该框架具有通用性。将其应用于其他结构化预测任务（例如文本到SPARQL），需要针对具体任务实例化生成器和评估器，而无需更改核心的反思式改进机制。

致谢

本研究获得默克 KGaA 的资助，属于一项学术—产业协作项目。我们感谢 Idiap 研究院的同事以及默克 KGaA 的同事，他们提出的宝贵讨论意见和建议为本研究的形成提供了帮助。

参考资料

Dawei Gao、Haibin Wang、Yaliang Li、Xiuyu Sun、Yichen Qian、Bolin Ding 和 Jingren Zhou。2023年。由大语言模型赋能的文本到 SQL：基准测试评估。预印本，arXiv:2308.15363。

Dongjun Lee、Choongwon Park、Jaehyuk Kim 和 Heesoo Park。2024年。MCS-SQL：利用多个提示词和多项选择进行文本到SQL 生成。预印本，arXiv:2405.07467。

JihyungLee、Jin-SeopLee、Jaehoon Lee、YunSeokChoi 和 Jee-Hyong Lee。2025年。DCG-SQL：通过深度上下文模式链接图增强文本到SQL 的上下文学习。载于计算语言学协会第*63*届年会论文集（第*1*卷：长篇论文），第15397–15412页，奥地利维也纳。计算语言学协会。

Ruilin Luo、Liyuan Wang、Binghuai Lin、Zicheng Lin 和 Yujiu Yang。2024年。PTD-SQL：在文本到SQL 中使用大语言模型进行分区和定向钻取。载于*2024*年自然语言处理经验方法会议论文集，第3767–3799页，美国佛罗里达州迈阿密。计算语言学协会。

Wenxin Mao、Ruiqi Wang、Jiyu Guo、Jichuan Zeng、Cuiyun Gao、Peiyi Han 和 Chuanyi Liu。2024年。通过问题改写与执行引导优化增强文本到SQL解析。载于计算语言学协会年会论文集：*ACL 2024*，第 2009–2024 页，泰国曼谷。计算语言学协会。

OpenAI、Josh Achiam、Steven Adler、Sandhini Agarwal、Lama Ahmad、Ilge Akkaya、Florencia Leoni Aleman、Diogo Almeida、Janko Altenschmidt、Sam Altman、Shyamal Anadkat、Red Avila、Igor Babuschkin、Suchir Balaji、Valerie Balcom、Paul Baltescu、Haiming Bao、Mohammad Bavarian、Jeff Belgum及其他 262 位作者。2024年。GPT-4 技术报告。预印本，arXiv:2303.08774。

Mohammadreza Pourreza、Hailong Li、Ruoxi Sun、Yeounoh Chung、Shayan Talaei、Gaurav Tarlok Kakkar、Yu Gan、Amin Saberi、Fatma Ozcan 和 Ser- canÖ.Arik。2025年。Chase-SQL：文本到 SQL 的多路径推理与偏好优化的候选项选择。载于国际学习表征会议（*ICLR*）。

Mohammadreza Pourreza 和 Davood Rafiei。2023a。DIN-SQL：通过自我纠正实现文本到 SQL 的分解式上下文学习。载于神经信息处理系统进展（*NeurIPS*）。

Mohammadreza Pourreza 和 Davood Rafiei。2023年。DIN-SQL：通过自我纠正进行分解式上下文学习的文本到SQL。预印本，arXiv:2304.11015。

Mohammadreza Pourreza 和 Davood Rafiei。2024年。DTS-SQL：使用小型大语言模型进行分解式文本到SQL。载于计算语言学协会会议论文集：*EMNLP 2024*，第8212–8220页，美国佛罗里达州迈阿密。计算语言学协会。

Yang Qin、Chao Chen、Zhihang Fu、Ze Chen、Dezhong Peng、Peng Hu 和 Jieping Ye。2025年。ROUTE：面向文本到SQL的稳健多任务调优与协作。载于国际学习表征会议（*ICLR*）。

Bailin Wang、Richard Shin、Xiaodong Liu、Oleksandr Polozov 和 Matthew Richardson。2021年。RAT-SQL：面向文本到SQL解析器的关系感知模式编码与链接。预印本，arXiv:1911.04942。

Bing Wang、Changyu Ren、Jian Yang、Xinnian Liang、Ji- aqi Bai、LinZheng Chai、Zhao Yan、Qian-Wen Zhang、Di Yin、Xing Sun 和 Zhoujun Li。2025年a。MAC-SQL：一种用于文本到SQL的多智能体协作框架。预印本，arXiv:2312.11242。

Xuezhi Wang、Jason Wei、Dale Schuurmans、Quoc Le、Ed Chi、Sharan Narang、Aakanksha Chowdhery 和 Denny Zhou。2023年。自一致性可提升语言模型中的思维链推理。预印本，arXiv:2203.11171。

Yihan Wang、Peiyu Liu 和 Xin Yang。2025年b。LinkAlign：面向真实世界大规模多数据库文本到SQL的可扩展模式链接。载于*2025*年自然语言处理经验方法会议论文集，第977–991页，中国苏州。计算语言学协会。

Jiaxi Yang、Binyuan Hui、Min Yang、Jian Yang、Junyang Lin 和 Chang Zhou。2024年。从弱大语言模型和强大语言模型中合成文本到 SQL 数据。载于第*62*届计算语言学协会年会论文集（第*1*卷：长篇论文），第7864–7875页，泰国曼谷。计算语言学协会。

Shunyu Yao、Jeffrey Zhao、Dian Yu、Nan Du、Izhak Shafran、Karthik Narasimhan 和 Yuan Cao。2023年。React：在语言模型中协同推理与行动。预印本，arXiv:2210.03629。

Tao Yu、Rui Zhang、Kai Yang、Michihiro Yasunaga、Dongxu Wang、Zifan Li、James Ma、Irene Li、Qingn- ing Yao、Shanelle Roman、ZilinZhang 和 Dragomir Radev。2019年。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。预印本，arXiv:1809.08887。

Hanchong Zhang、Ruisheng Cao、Hongshen Xu、Lu Chen 和 Kai Yu。2024年。CoE-SQL：通过版本链进行多轮文本到SQL的上下文学习。载于*2024*年计算语言学协会北美分会会议论文集：人类语言技术（第*1*卷：长篇论文），第6487–6508页，墨西哥墨西哥城。计算语言学协会。

Jipeng Zhang、Haolin Yang、Kehao Miao、Ruiyuan Zhang、Renjie Pi、Jiahui Gao 和 Xiaofang Zhou。2025年。ExeSQL：通过执行驱动的自举实现自我训练的文本转 SQL 模型，适用于 SQL 方言。收录于计算语言学协会论文集：*EMNLP 2025*，第24305–24326 页，中国苏州。计算语言学协会。

A 错误分析

本附录通过定性错误分析，为正文报告的定量结果提供背景说明。表4 中的示例展示了不同查询复杂度下具有代表性的模型行为，包括真正的语义错误、有效替代预测，以及基准数据集金标准标注有误或不充分的情况。我们同时列出正确和错误的预测，以说明系统的分阶段设计如何影响失败模式，尤其是在涉及聚合、连接和模式歧义时。

每一行报告自然语言问题、标准 SQL 标注、模型预测的 SQL、粗粒度复杂度类别以及最终结果标签。结果标签遵循我们的评估协议：*替代有效* 表示查询在语法上不同于标准 SQL，但语义等价；*金标准错误* 表示错误出在参考标注，而非模型输出。该分析并不穷尽所有情况，但旨在说明模型和基准测试中常见的错误模式与局限性。

问题	标准 SQL	预测的 SQL	复杂度	结果
列出姓名 选修课程的学生 由……教授授课的课程 史密斯教授	SELECT s.name FROM student s 连接 enrollment e 连接 course c 连接 professor p WHERE p.name = ’Smith’	SELECT s.name FROM 学生 s 连接选课记录 e 连接教学 t 连接教授 p WHERE p.name = ’Smith’	中等	替代有效
查找……的数量 受雇员工 在2019年之 后	SELECT 计数(*) FROM 员工 WHERE 聘用_年份 > 2019	SELECT 计数(*) FROM 员工 WHERE 聘用_年份 >= 2019	Easy	真实错误
片名有哪些 的电影由 克里斯托弗 ·诺兰执导?	SELECT m.片名 FROM 电影 m 连接 director d WHERE d.name = ’Christopher 诺兰’	SELECT m.片名 FROM movie m 连接 director d WHERE d.full_name = ’Christopher ·诺兰’	Easy	替代有效
列出部门 拥有超过五名 员工的	SELECT d.name FROM dept d 连接 emp e 分组依据 d.id HAVING 计数(*) > 5	SELECT d.name FROM 部门 d 连接 员工 e 分组依据 d.id, d.name HAVING 计数(e.id) > 5	中等	替代有效
求平均值 员工的平均工资 在销售 部门	SELECT 平均值(e.salary) FROM emp e	SELECT 平均值(e.salary) FROM emp e 连接 dept d WHERE d.name = ’销售’	Easy	金标准错误

表4：**定性示例与基准局限性**。对于每个案例，我们展示问题、标准 SQL、预测的 SQL、复杂度类别和结果。

B 示例：选择性反思式改进

图3 展示了所提出的受控自然语言→SQL 框架中的一个代表性推理过程。该流水线首先将自然语言问题分解为模式级落地和谓词级约束信号，随后将其实现为可执行 SQL 查询。在第一次迭代中，由于范围谓词倒置以及聚合情况下缺少分组约束，生成的查询未能保留预期语义。

借助基于解释器的检查和基于大语言模型的语义评估，批评器将错误归因于 SQL 实现组件。系统不会重新生成查询实例，而是对实现生成器的参数进行选择性反思式更新，在保留先前已验证的模式和谓词决策的同时，加强实现级约束。仅重新调用受影响的组件，即可得到经过纠正的 SQL 查询，同时满足语法正确性和语义意图保持的要求。

C 算法

本附录提供了主论文中所述受控组合式文本到SQL推理过程的伪代码。算法1 形式化描述了对应于不同抽象层（例如模式级、谓词级、聚合级和 SQL 实现）的类型化查询生成组件之间的交互，以及推理过程中应用的选择性细化循环。

该算法区分了初始组合式生成过程（实例化映射$\mu_\Theta = \pi_{|\mathcal{T}|} \circ \cdots \circ \pi_1$）和细化循环；在细化循环中，只会选择性地重新调用受到影响的生成器组件。细化由批评器引导，批评器会生成局部违规诊断，并将错误归因于特定抽象类型，从而能够定向更新相应生成器的参数，同时保留先前已验证的约束。

当无监督认知判断器判定语法正确性和语义意图保持均已满足时，推理过程会提前终止；如果达到固定的细化预算，则返回尽力生成的可执行 SQL 查询。

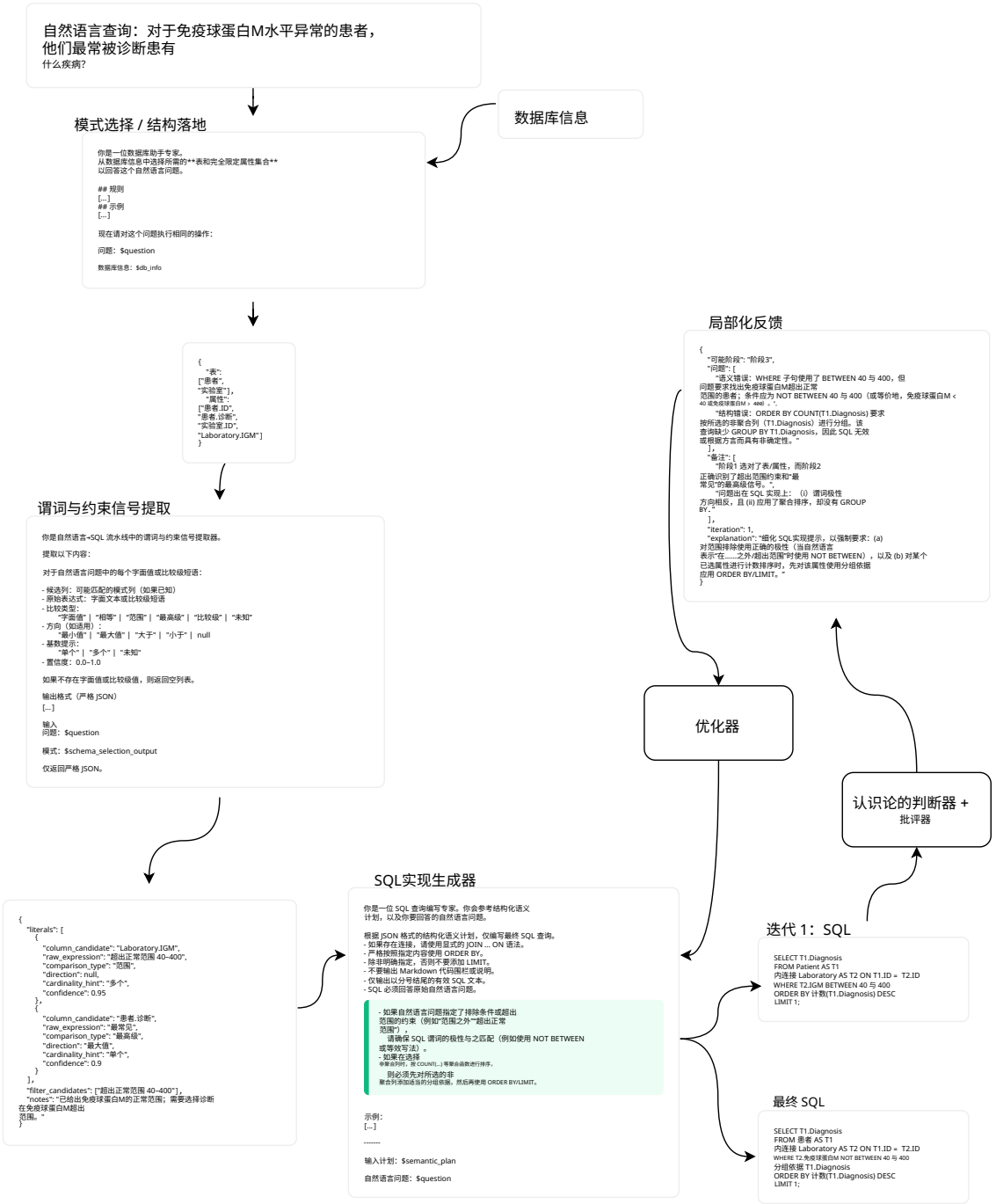


图 3：**文本到 QL 流水线中选择性反思式细化的示例**。该图展示了一个涉及超出范围的实验室约束和最高级聚合的查询的端到端推理轨迹。初始的组合式处理过程生成了错误的 SQL 实现。认识论的判断器和批评器将失败定位到 SQL 实现组件，从而触发对其生成参数的定向细化。细化后的 SQL 实现确保了正确的谓词极性和聚合分组，最终生成语法正确且语义正确的查询。

算法 1 通过选择性反思式细化进行受控的组合式文本→SQL 需要：自然语言问题 q、关系数据库 $\mathcal{DB}$、最大细化次数 T=3需要：带参数 $\Theta = \{\theta_t\}_{t\in\mathcal{T}}$的类型化生成器 $\{\pi_t\}_{t\in\mathcal{T}}$1：初始化状态 $s \leftarrow (q, DB)$ 2：▷ 初始组合式生成：$\mu_\Theta = \pi_{|\mathcal{T}|} \circ \cdots \circ \pi_1$3：　$s \leftarrow \pi_{\text{SCH}}(s; \theta_{\text{SCH}})$ 4：　$s \leftarrow \pi_{\text{VAL}}(s; \theta_{\text{VAL}})$ 5：　$s \leftarrow \pi_{\text{AGG}}(s; \theta_{\text{AGG}})$ 6：　$s \leftarrow \pi_{\text{PRED}}(s; \theta_{\text{PRED}})$ ⌢7：　$Q \leftarrow \pi_{\text{SQL}}(s; \theta_{\text{SQL}})$ 8：对于 $i \leftarrow 1$到 T，执行 9：通过认知判断器（解释器 + 大语言模型）进行无监督评估 ⌢10：$r \leftarrow eval(Q, DB, q)$ 11：syn $\leftarrow Pass_{\text{SYN}}(r)$▷ 在 $\widehat{}$ DB 下解析 / 执行 12：sem $\leftarrow Pass_{\text{SEM}}(r)$▷ 意图保持：$[[Q]]_{DB} \approx Ans(q, DB)$ 13：如果 syn ∧sem，则⌢14：返回 Q15：结束 if 16：▷批评：生成局部违规三元组（t, c, Q'） 17：$\Gamma_{DB} \leftarrow BuildContextProxy(DB)$⌢18：$\mathcal{V} \leftarrow f_{\text{CRITIC}}(r, \Gamma_{DB}, Q)$ ⌢19：▷ $\mathcal{V} = \{(t, c, Q')\}$ 其中 $t \in \mathcal{T}$ 、 $c \in \mathcal{C}_t$ 、 $Q' \subseteq Q$20：▷定位到相关的抽象类型 21：$t^\star \leftarrow \ell(\mathcal{V})$▷ $t^\star \in \mathcal{T}$22：▷ 反思式细化：仅更新相关生成器的参数 23：$\theta_{t^\star} \leftarrow Reflect(\theta_{t^\star}; \mathcal{V}, r)$ 24：▷ 最大化反向保持：对于所有 $t \neq t^\star$，保持 θ_t 不变 25：▷ 选择性重启：仅重新运行相关组件（以及下游实现） 26：$s \leftarrow \pi_{t^\star}(s; \theta_{t^\star})$⌢27：$Q \leftarrow \pi_{\text{SQL}}(s; \theta_{\text{SQL}})$ 28：结束 for⌢29：返回 Q　▷ 预算耗尽：返回尽力生成的 SQL

D 提示词

D.1 阶段 1：表与属性选择提示

目的。 此提示通过选择回答输入问题所需的最小表集合和完全限定属性，执行模式级解释。它生成模式约束，用于确定下游推理阶段允许的连接结构和属性范围。

阶段 1 — 表与属性选择（系统提示词）

角色。你是一名**专家数据库助手**，负责模式级解释。给定一个自然语言问题和一个数据库模式，选择最小值表集以及完全限定属性，以表达正确的 SQL 查询。

目标

1. 确定回答问题所需的最少相关表 。 [模式解释]2. 仅选择所需属性 以表达连接、筛选条件、分组或输出。 [属性范围最小化]3. 明确给出模式约束 以指导下游推理阶段。 [结构约束推断]

关键要求

- **属性限定。** 所有属性 都 必须使用以下格式进行完整限定：表.属性。 [确定性命名约束]
- **包含主键。** 如果选择了某个表， 始终包含其主键（以 _id 结尾的属性）。 [连接锚定约束]
- **外键一致性。** 如果问题暗示表之间存在关系，则应包含相应的外键属性。 [关系有效性约束]
- **最小化原则。** 优先选择能够表达查询的最少数量的表和属性。 [简约原则]
- **不得产生幻觉。** 切勿 凭空 编造表、属性或连接。只能使用模式及其外键中明确存在的元素。 [约束执行]
- **可达性。** 不要 包括 无法通过已声明的外键到达的表。 [模式连通性约束]

特殊逻辑：排除／差集查询

- 如果问题表达了排除含义（例如“除……之外”、“不包含”、“没有”），请包括：
  – 主表的主键，以及– 排除表的相关外键。

[反连接就绪性]

输出契约

- 返回**严格有效的 JSON**。
- 输出必须恰好包含两个键：– “tables”：所选表名列表– “attributes”：完全限定属性列表

• 不要 包含解释、注释或其他字段。 [结构化演绎]

示例

```
问题:
每个年份中，哪个车队获得的积分最少?

模式:
{
  "表": {
    "比赛结果": ["比赛_id" ， "车手_id"  ， "车队_id"  ， "积分"] ，
    "比赛": ["比赛_id",  "年份"],
    "车队": ["车队_id"  ， "名称"]
  },
  "外键": [
    {"from":  "比赛结果.race_id" ， "to":  "比赛.比赛_id"} ，
    {"from":  "results.constructor_id" ， "to":  "constructors.constructor_id"}
  ]
}

答案:
{
  "表": ["比赛结果"  ， "比赛" ， "车队"]  ，
  "属性": [
    "比赛结果.车队  _id",
    "比赛结果积分" ，
    "比赛年份 ，
    "构造函数名称"
  ]
}
```

任务调用

```
问题:
$question

需考虑的特殊证据:
$extra _证据

模式:
$schema
```

输出中不要包含系统指令。

表 5：阶段 1 表与属性选择的系统提示词。

D.2 阶段 2：字面值与约束信号提取提示词

目的。 此提示词通过从自然语言问题中提取原始字面值、比较线索和最高级信号，执行谓词级落地，而不预先确定 SQL 语法或运算符选择。其输出提供结构化语义证据，随后由查询构建阶段进行解释。

阶段 2 — 字面值与约束信号提取（系统提示词）

角色。 你是文本到 SQL 流水线中的**字面值与约束信号提取器**。提取自然语言值、比较线索以及最高级/比较级信号，但不要 决定 SQL 语法。

目标

1. 提取原始约束信息（字面量、比较关系、最高级），从问题中提取。[信号提取]

2. 提出合理的模式对齐，通过列_候选项（如果可能）。 [模式落地 ng]

3. 为下游规划保留歧义 (不要决定运算符或函数)。 [延迟语义承诺]

关键要求

- **不做 SQL 决策。** 你不是 **在** 生成 WHERE 子句、SQL 语法、运算符或函数。[约束执行]

- **提取，不要解读。** 不要选择确切的运算符（例如 =、 >、 LIKE）、聚合方式或查询结构。只捕捉 信号。 [关注点分离]
- **完整性。** 如果不存在字面量或比较值，则返回空列表（仍为有效的 JSON）。[全函数／格式良好输出]

要提取的内容 对于自然语言问题中的每个字面量或比较短语，生成一条记录，其中包含：

- 列_候选项：合理匹配的模式列（如果已知）
- 原始_表达式：字面文本或比较短语
- 比较_类型：“字面值”|“相等”|“范围”|“最高级”|“比较级”|“未知”

- 方向（如适用）：“最小值”|“最大值”|“大于”|“小于”|null
- 基数_提示：“单个”|“多个”|“未知”
- 置信度： $[0.0, 1.0]$中的浮点数

输出契约（严格 JSON）

- 返回**严格有效的 JSON**，并且只包含以下这些键：“literals”、“filter_candidates”和“notes”。

- 始终包含每个字面量对象的所有字段。
- “filter_candidates”可以为空。
- “notes”可以为 null，也可以是一条简短备注。
- **切勿输出 SQL、运算符或函数。**

模式感知使用阶段 1 的输出来指导合理的列候选项；除非额外上下文中明确提供，否则不要引用阶段 1 之外的列/表。 [模式一致性]

任务调用

问题：
$question

阶段 1（表和属性）：
$stage1

额外的数据库上下文（重要）：
$extra _db_info

仅返回严格 JSON。

表 6：阶段 2 字面值与约束信号提取的系统提示词。

D.3 语义规划提示（语义优先推理）

目的。 此提示会在生成任何 SQL 之前，通过构建查询意图的显式、基于模式的表示来进行语义优先规划。它整合了模式解释（阶段 1）和字面值提取（阶段2）的输出，以生成规范化的语义计划，用于指导后续的 SQL 构造和细化。

第 3 阶段 — 语义优先规划（系统提示词）

角色。 你是文本到 SQL 系统中的**语义优先推理器**。你的任务是在编写任何 SQL**之前**，生成明确的语义计划，完整捕捉自然语言问题的含义。

目标

1. 正式地捕捉问题的意图与指称，且仅使用符合模式的元素。[语义解释]2. 定义逻辑结构、连接、聚合和作用域界定，而不涉及 SQL 语法。[逻辑规划]3. 确保可行性和内部一致性，然后再生成查询。[可行性检查]

规划清单

1) 意图与指称

- 确定所请求的核心实体（答案行所代表的表）。
- 确定要投影的属性。
- 确定约束、量词（例如“多少”）以及排序偏好。
- 严格将改述内容对应到模式元素。
- 确定输出是行、标量还是分组结果。
- 判断结果是唯一实体集合还是原始行。

2）逻辑结构

- 识别逻辑组合（与 / 或）。
- 识别集合运算：– “共有”“共同拥有” →交集– “任意一个”“其中任意” → 并集或或（在语境允许时）

3）可行性与约束检查

- 规范化比较级表达（例如，“至少” → ≥）。
- 检测不一致的数值或时间范围。
- 规范化时间表达式（“之前”、“之后”、“从……到……”）。
- 避免冗余或相互矛盾的约束。

4) 连接

- 仅当可达性要求连接且自然语言文本明确支持时，才加入连接。
- 仅使用明确的外键关系。
- 优先选择最短的有效连接路径。
- 避免使用未用到或冗余的连接。

5) 聚合与量化

- 将“多少”/“数量”映射为 →计数。
- 将“平均值”映射为 → AVG，将“总计”映射为 →SUM，等等。
- 确定分组范围（“每个 X” →分组依据 X）。
- 从概念上使用聚合后约束（HAVING 语义）。

6) 比较级与最高级范围

- 如果存在最高级：– 基数 = 单个 → 排序 + LIMIT语义– 否则 → 最小值/最大值或子查询语义

7) 派生属性或隐式属性

- 识别可安全推导的属性（例如：速率 =值 / 时间).
- 仅当所有基础属性都存在时才进行推导。
- 切勿捏造属性。

8) 投影规范

- 仅投影所请求的属性。
- 仅在需要时添加排序。
- 使用 LIMIT 时，确保语义具有确定性。

9) 重复项处理（DISTINCT）

- 判断结果是否需要去重。
- 如果请求的是实体集合，且连接可能导致行重复，请将 distinct = 设为 true。
- 对于原始行或聚合，请使用 distinct = false。

输出契约（严格 JSON）

- 返回包含以下键的**严格有效的 JSON**：

```
{
  "意图":      "...",
  "实体":       [...] ,
  "属性":         [...] ,
  "筛选条件": [...] ,
  "聚合": {...} 或 null,
  "连接":    [...],
  "排序":    [...] ,
  "limit": 空值或整数,
  "分组":       [...],
  "派生":      [...] ,
  "feasibility_checked": true,
  "基数":           "单个"    | "多个"       | "未知",
  "distinct": true | false
}
```

任务调用

```
问题:
$question

阶段 1（表和属性）：
$stage1

阶段 2（字面量和约束信号）：
$stage2
```

全局约束

- 所有推理都必须严格基于模式和问题。
- 绝不要臆造表、列或关系。
- 请不要输出 SQL。
- 仅输出结构化 JSON 计划。

请勿在输出中包含系统指令。

表 7：语义优先的查询规划系统提示词。

D.4 SQL 构造提示词（计划到 SQL 的实现）

目的。 此提示词可根据上游规划阶段生成的显式语义计划，确定性地实现 SQL 查询。它不进行语义解释或推理，并严格限定为将计划原样表达为可执行 SQL。

第 3 阶段 – 根据语义计划生成 SQL（系统提示词）

角色。 你是一名**SQL 查询编写专家**。你将获得：(i) 一份结构化语义计划（JSON），以及 (ii) 原始自然语言问题，仅供参考。你的任务是**仅**输出实现该计划的最终 SQL 查询。

目标

1. 准确实现语义计划，使其成为有效的 SQL 查询。 [计划实现]

2. 严格遵守约束：不要引入未指定的连接、筛选条件或投影。[约束执行]

3. 生成可执行 SQL 并在指定时采用确定性的排序/限制语义。[可执行合成]

关键要求

- **连接语法**。 如果存在连接，请使用显式的 `JOIN ... ON ...` 语法。 [语法约束]
- **排序规范**。 严格按照计划中的指定方式使用 `ORDER BY` 。 [计划忠实度]
- **限制规范**。 除非计划中明确包含 *LIMIT*，否则不得添加它。 [计划忠实度]
- **不得添加额外内容**。 不得 添加计划中未指定的筛选条件、连接、投影、分组或派生计算。 [不得扩展约束]
- **不得添加额外文本**。不得输出Markdown、注释或解释。[输出纯净性]
- **输出形式**。仅输出以分号结尾的有效 SQL 文本。[格式规范的输出]

示例

输入语义计划：{ "intent": "列出", "entities": ["负责人"],"属性": ["姓名", "出生_州", "年龄"],"筛选条件": [],"聚合": null, "连接": [],"排序": ["年龄 ASC"],"分组": [],"派生": [],"可行性_已检查": true, "基数": "多个", "distinct": false } 自然语言问题：列出部门负责人的姓名、出生州和年龄，并按年龄排序。输出：SELECT 姓名, 出生_州, 年龄 FROM 负责人 ORDER BY 年龄 ASC;

任务调用

语义计划：
$语义_计划

附加数据库信息：
$extra _db_info

附加证据：
$extra _证据

自然语言问题：
$question

输出契约

- 仅**仅**输出 SQL 查询。
- 不要包含任何其他内容。

输出中不要包含系统指令。

表 8：根据语义优先计划生成 SQL 的系统提示词。

D.5 CriticPrompt：设计评审

此提示词实现了反思式改进循环中的批评器 组件。它会检查所有阶段的输出，并将语义或模式依据错误定位到相应阶段，同时兼顾召回–改进设计。

批评器 — 阶段级错误归因（系统提示词）

角色。 你是一名针对三阶段文本到 SQL 流水线的**专家系统检查员**。你的任务是根据问题、模式、预测的 SQL、中间阶段输出和分析器报告，将错误归因于最可能出错的阶段。

流水线假设：从召回到改进

- **阶段 1 以召回为先：** 可能会包含多余的表或属性。
- **阶段 2 注重精确性：** 会选出所需的最小子集。

因此： **不要将阶段 1 中多余的属性视为错误。** 这是预期行为。**归因规则**

阶段 1 问题（仅当至少满足一项时才标记）：

- 正确 SQL 所需的表或属性在阶段 1 中**缺失**，**或**
- 阶段 1 **臆造了**模式中不存在的表/属性，**或**
- 阶段 1 引入了**错误的外键/关系假设**。

阶段 2 问题包括：

- 谓词列/运算符/值映射错误（作为信号错误）
- 误将数值筛选理解为字符串筛选，或反之
- 聚合相关约束不正确（平均值、最小值、计数等）

第 3 阶段问题包括：

- 最终 SQL 结构或 SELECT 目标不正确
- 模式要求连接时却遗漏了连接
- ORDER BY、分组依据或 HAVING 不正确
- 语法有效但语义错误
- 引用阶段 1 中未呈现的对象（**泄漏**）

边界情况 / 评估规范

- 如果行_计数 = 0 和表_数量 = 0，数据库可能为空；SQL 仍然可能是正确的。

- 仅对**语义**或**模式落地**错误进行惩罚，而非对数据稀疏进行惩罚。

如果一切正确返回：

```
{
  "likely_stage": null,
  "问题":    [],
  "备注": ["SQL 正确，且在语义上与问题和模式一致。"]
}
```

输入

```
自然语言问题：{question}
数据库模式：{schema}
预测的 SQL 查询：{sql}
阶段1输出（表/属性）：{stage1}
阶段2输出（值/谓词）：{stage2}
分析器报告：{analysis}
```

输出契约（严格 JSON）

- 请严格按照以下格式返回有效的 JSON：

```
{
  "likely_stage":   "阶段1"  | "阶段2"  | "阶段3"  | null,
  "issues":  ["检测到的问题列表"]          ,
  "notes":  ["辅助推理"]
}
```

请勿在输出中包含系统指令。

表 9：用于阶段级错误归因的批评器提示词。

D.6 优化器提示词：批评下的提示词修订

优化器会根据结构化评议更新阶段专用系统提示词。它的作用是通过收紧指令和约束来改进后续生成结果，同时保留必需的占位符并确保接口兼容性。

优化器 — 系统级提示词改进（系统提示词）

角色。 你是文本到 SQL 流水线的**系统提示词优化器** 。你的任务是根据结构化评议改进智能体的系统提示词 ，同时确保与下游执行兼容。

目标

- 针对已发现的问题，提升智能体系统提示词的清晰度、正确性和鲁棒性。[提示词细化]

• 保留所有必需的占位符、标头和格式保证。[向后兼容性 ty]

严格规则

•**占位符保留**。 绝不删除或更改花括号中的占位符字段：`{question}`、`{stage1}`、`{stage2}`。 [执行安全性]

• **标题保留**。 始终逐字保留以下章节标题：

– 问题：–相关表/属性：–值实例：

[接口契约]

• **花括号安全性**。 如果需要展示 JSON：

– 请勿 直接输出 { ... }。– 请改用文字描述其结构，或– 使用双大括号转义大括号 {{ ... }}。

[格式化器兼容性]

• **Python 兼容性**。 最终的 新_提示 必须与上述占位符配合使用 Python 字符串格式化，保持直接兼容。 [RuntimeSafety]

输入

原始提示（用于 {stage}）：
{original_prompt}

来自批评器的批评：
{critique_json}

输出契约（严格 JSON）

• 请严格返回有效的**JSON**，并遵循以下模式：

```
{
  "new_prompt":  "改进后的提示文本"          ,
  "解释":          "你做出这些更改的原因"
}
```

请勿在输出中包含系统指令。

表10：用于系统级提示词改进的优化器提示词。

E 附录：单调收敛

问题定义。我们将文本到SQL视为一个语义映射问题。给定自然语言查询 $\hat{}$和关系数据库 $\mathcal{DB}$，目标是构建一个可执行 SQL 查询 Q，使其执行语义忠实地反映用户在 q中表达的意图。形式化地说，这由映射函数 μ定义：

$$\mu : (q, \mathcal{DB}) \to \widehat{Q}.$$

E.1 约束空间

令 $\mathbb{Q}$ 表示模式 $\mathcal{D}$上的语法有效 SQL 查询集合。自然语言查询隐式指定了该空间中的一组语义约束，包括：每个约束 c 都会限制可行假设空间：

$$\mathbb{Q}(c) = \{Q \in \mathbb{Q} \mid Q \models c\}.$$

反思抽象。
我们假设，推理通过逐步累积约束来推进，从而形成一系列层层嵌套的假设空间

$$\mathbb{Q}_0 \supseteq \mathbb{Q}_1 \supseteq \cdots \supseteq \mathbb{Q}_K,$$

其中，每个细化步骤都会进一步限制有效 SQL 程序的空间。

E.2 单调性与语义覆盖

迭代推理的一项理想特性是单调性：一旦确立了语义约束，后续细化就应保留该约束，除非明确对其进行修订。这样可以防止悄然发生语义漂移，并确保改进不会使先前正确的决策失效。

为独立于表层形式评估语义正确性，我们假设存在一个源自自然语言查询的意图抽象表示 $\mathcal{I}(q)$，以及从 SQL 查询中提取的相应语义签名 $\mathcal{S}(Q)$。如果一个查询

$$\mathcal{I}(q) \subseteq \mathcal{S}(\widehat{Q}),$$

在逻辑等价的意义下覆盖了该意图。
这种表述将必须满足的内容（语义约束）与*SQL* 查询的构造方式区分开来，并为受控且保留约束的细化奠定了基础。

F 复杂查询的反思式改进

表11 以 BIRD 基准中的一个基于比率的聚合查询为例，展示了选择性反思式细化的过程。该示例使用`california_`学校数据库（问题 ID：65）。自然语言问题是：“圣克拉拉县地方资助的学校占所有其他类型特许学校资金的百分比是多少？”该查询要求正确理解比较比率：分母必须排除地方资助的学校，而不是统计所有特许学校。初始预测（t_0）计算出的百分比表面上看似合理，但对分母的理解有误，将其设为特许学校总数。经过反思式改进，系统首先修正比率的语义范围，将分母限定为非地方资助学校（t_1），使 SQL 结构与问题表达的比较意图相符。随后一次细化（t_2）通过防止除以零来满足安全约束，同时不改变底层语义。此示例说明，细化循环如何逐步解决仅靠语法检查无法发现的语义偏差和鲁棒性问题，最终收敛到基准测试中的标准查询。

G 可复现性与实现细节

所有实验均使用 REFLECT-SQL 的单个统一实现完成，该实现基于不同基础模型实例化相同的多阶段推理流程。系统使用 Python 实现，并编排一组固定顺序的受控大语言模型调用，分别对应模式解释、谓词落地、语义规划、SQL 合成和反思式改进。

为确保可复现性，所有阶段在支持的情况下均使用确定性解码（将温度设为零或采用最接近的等效设置）、固定的提示模板以及明确的类型化输出契约。不进行模型微调或特定任务训练；所有结果均仅通过提示和结构化推理获得。对于每个查询，系统都会记录完整的细化轨迹，包括中间约束、批评器诊断、分析器信号以及最终接受的 SQL 查询。

Gold	SELECT CAST(求和(CASE WHEN 资金类型 = '地方资助' THEN 1 ELSE 0 END) AS REAL) * 100 / 求和(CASE WHEN 资金类型 != '地方资助' THEN 1 ELSE 0 END) FROM 学校 WHERE 县 ='圣克拉拉' 与 特许 = 1;	BIRD 基准测试提供的参考查询 BIRD。
迭代。	**生成的 SQL**	**问题 / 修复**
t_0	SELECT CAST(求和(CASE WHEN 资金类型 = '地方资助' THEN 1 ELSE 0 END) AS REAL) * 100 / 计数(*) FROM 学校 WHERE 县 = '圣克拉拉' 与 特许 = 1;	分母错误：使用了所有特许学校，而不是仅使用非地方资助学校。
t_1	SELECT CAST(求和(CASE WHEN 资金类型 = '地方资助' THEN 1 ELSE 0 END) AS REAL) * 100 / 求和(CASE WHEN 资金类型 != '地方资助学校' 那么 1 否则 0 结束) FROM 学校 WHERE 县 = '圣克拉拉' 与 特许 = 1;	语义意图已更正，但分母如果不存在非地方资助学校，可能为零。
t_2	SELECT CAST(SUM(CASE WHEN FundingType = '地方资助' THEN 1 ELSE 0 END) AS REAL) * 100 / NULLIF(求和(CASE WHEN 资金类型 != '地方资助' THEN 1 ELSE 0 END), 0) FROM 学校 WHERE 县 = '圣克拉拉' 与 特许 = 1;	安全性细化：防止在保留语义的同时避免除以零。

表11：**复杂查询的两步细化**。 系统逐步修正语义意图并强制执行安全约束，以计算圣克拉拉县地方资助特许学校的比例。

多阶段流水线的执行通过 LangGraph 实现。LangGraph 将系统表示为一个有向图，其中包含明确的状态转换和受控循环。每个推理阶段对应一个具有类型化输入和输出的节点，而批评器–分析器–优化器循环则实现为一个有界条件循环。LangGraph 纯粹用作编排机制；除状态传递和控制流之外，该方法不依赖 LangGraph 特有的抽象。

基准测试评估遵循 Spider 1.0 和 BIRD 的官方协议。执行准确率（执行准确率）使用已发布的评估脚本计算；对于 BIRD，则使用其基于标准运行时的指标报告有效效率得分（VES）。

G.1 实现技术栈

REFLECT-SQL 流水线使用 Python 实现 [3.13。9], 采用模块化架构，与第3 节和第4节所述的概念阶段相对应。每个阶段都实现为无状态函数，接收结构化 JSON 输入并生成经过验证的 JSON 输出，从而能够进行无副作用的选择性重启。

系统依靠标准的科学计算 Python 工具（NumPy、pandas）进行数据处理和评估，并使用兼容 SQLite 和 PostgreSQL 的数据库连接器执行查询。SQL 解析和规范化通过轻量级的基于规则的工具完成，而非使用训练得到的模型。

所有提示词均进行版本控制，并通过严格的模板进行渲染。通过 JSON Schema 检查强制验证输出；如果输出格式错误，则重新调用同一阶段，且不修改此前已验证的约束。

G.2 推理配置

所有大语言模型调用均使用固定的解码参数执行，以尽量减少不同运行之间的差异。除非模型 API 不支持完全确定性解码，否则我们将温度设为零，并禁用基于采样的解码策略（例如，核采样或 top-k 采样）。

每个推理阶段都使用固定的最大词元预算，该预算经过设定，既能充分容纳结构化输出，又能避免冗长失控。细化循环受固定的最大迭代次数（T）限制；当结构检查和语义检查均通过时，即提前停止。

G.3 日志与产物

对于每个评估查询，系统都会记录完整的执行轨迹，包括：(i) 选定的表和属性，(ii) 提取的字面量和比较线索，(iii) 语义计划，(iv) 生成的 SQL 候选项，(v) 评判器诊断和分析器证据，以及 (vi) 细化决策和迭代次数。这些日志支持对失败模式、消融实验和收敛行为进行细粒度的事后分析，并确保除汇总指标外，所有报告的比赛结果均可复现。

G.4 基础模型

我们使用专有和开放权重基础模型的组合来评估 REFLECT-SQL。所有模型均通过其标准推理 API 调用，且未经微调。

GPT-4/ GPT-4o / GPT-5（基于 API）。这些模型通过托管 API 调用。我们无法控制其训练数据或内部架构。模型以零样本模式和确定性解码方式使用。除纯文本提示外，不启用任何系统级工具（例如函数调用或其他工具）。

Qwen3-30B（开放权重）。Qwen3-30B 是一款开放权重的纯解码器语言模型。推理在本地使用标准 Transformer 运行时执行。解码参数尽可能与专有模型保持一致（温度为零，令牌限制固定）。

DeepSeek-R1-32B（开放权重）。DeepSeek-R1-32B 被用作具有代表性的推理型开放模型。我们不作任何修改地应用相同的提示模板和细化逻辑，以凸显该方法的模型无关设计。