arXiv:2602.07371v1 [cs.DB] 7 Feb 2026

DeepPrep：一种用于自主数据准备的大语言模型驱动的智能体系统

MeihaoFan†, Ju Fan†, YuxinZhang†, Shaolei Zhang†, Xiaoyong Du†,
Jie Song*, Peng Li*, Fuxin Jiang*, Tieying Zhang*, Jianjun Chen*
† 中国人民大学； * 字节跳动；
{fmh1art,fanj,yuxin.zhang,zhangshaolei98,duyong}@ruc.edu.cn
{jie.song,peng.li01,jiangfuxin,tieying.zhang,jianjun.chen}@bytedance.com

摘要

数据准备旨在将异构且含噪的原始表转换为可供分析的数据，仍然是数据科学中的一大瓶颈。近期方法借助大语言模型（LLM），根据自然语言规范自动完成数据准备。然而，现有的大语言模型驱动方法要么在缺乏中间执行结果依据的情况下做出决策，要么依赖线性交互过程，难以充分支持对先前决策的修订。为解决这些局限，我们提出 DeepPRep，一种用于自主数据准备的大语言模型驱动的智能体系统。DeepPRep 通过与环境进行迭代式、基于执行的交互来构建数据准备流水线；该环境会生成中间表状态并返回运行时反馈。为克服线性交互的局限，DeepPRep 采用基于树的智能体推理来组织流水线构建，从而能够基于执行反馈开展结构化探索和非局部修订。为有效学习此类行为，我们提出渐进式智能体训练框架，并结合数据合成来提供多样且复杂的 ADP任务。大量实验表明，DeepPRep 的数据准备准确率可媲美强大的闭源模型（例如，GPT-5），且推理成本更低；同时，它在开源基线中取得了最先进的性能，并能有效泛化到多样化的数据集。

PVLDB 引用格式：
作者。DeepPRep：一种用于自主数据准备的大语言模型驱动的智能体系统。PVLDB，19(1)：XXX-XXX，2026年。
doi:XX.XX/XXX.XX

PVLDB 制品可用性：
源代码、数据和/或其他制品已在 xxxx 提供。

1 引言

数据准备仍然是数据科学中的一大瓶颈，通常会占用端到端分析时间的 60%–80% [6]。这是由于不同来源之间存在异质性、噪声和不一致性，同时数据准备还具有迭代性和相互依赖性

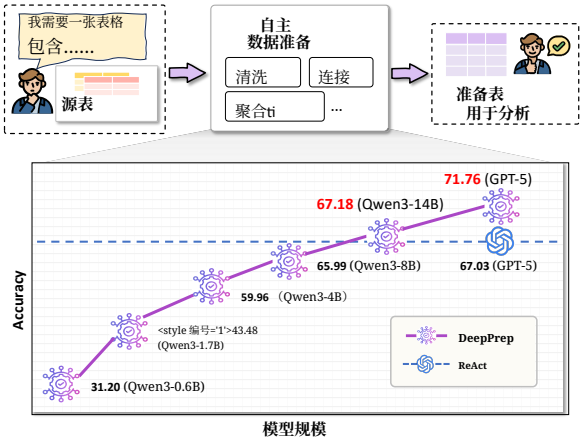


图 1：自主数据准备概览（上）及 DeepPRep 的性能（下）。

等清洗、连接和聚合等操作。为应对这些挑战，自主数据准备（ADP）旨在根据高层级目标规格，通过生成可执行的转换流水线，自动将多个源表转换为可供分析的数据。例如，如图 1 所示，给定多个源表和类似“我需要一张包含……的表”的目标描述，系统会自动生成并执行所需的转换步骤。

最先进的方法。 现有的自主数据准备方法大致分为两类。

- 传统方法依赖严格的规范，例如输入–输出示例、转换模板或基于规则的程序合成 [3, 16, 17, 19, 35]。这些方法在受限场景中效果良好，但并非为灵活适应以自然语言表达的数据准备意图而设计。
- 近期方法利用大语言模型（LLM）根据自然语言规范自动完成数据准备 [7,29, 31]。基于提示的方法直接根据任务描述和示例生成数据准备代码[23, 28, 32]，但其决策缺乏对中间执行结果的系统性依据，常常导致处理流水线容易出错。ReAct 风格的方法通过交替进行推理与特定任务的动作来纳入执行反馈 [2, 41]。然而，这种交互沿着线性轨迹进行，无法明确支持修正早先的决策；对于自主数据准备而言，这可能并不足够，因为早期错误往往要经过若干依赖操作后才会显现。

本作品依据知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议授权。请访问 https://creativecommons.org/licenses/by-nc-nd/4.0/ 查看许可协议副本。如需进行超出许可协议范围的使用，请发送邮件至 info@vldb.org 获取许可。版权所有者为作品所有者/作者。出版权已授予 VLDB 基金会。VLDB 基金会会刊，第 19 卷，第 1 期，ISSN 2150-8097。doi:XX.XX/XXX.XX

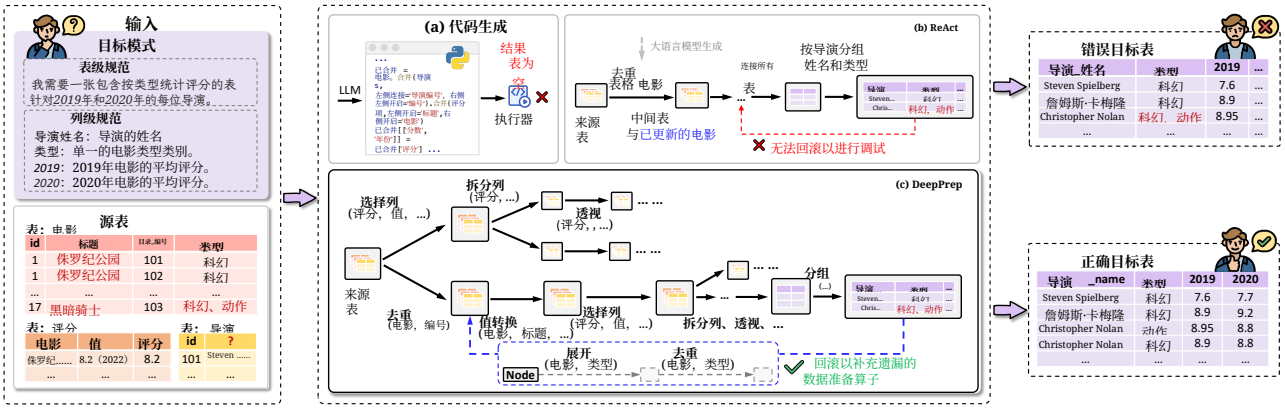


图2：一个展示 DeepPRep 核心思想的示例。(a) 代码生成在没有中间执行结果作为依据的情况下做出决策。(b) ReAct 风格方法遵循线性推理，修正先前决策的能力有限。(c) DeepPRep 采用基于树的结构化探索，从而能够通过回溯解决非局部错误。

我们的方案。 为解决这些局限，我们提出了DEEP-PREP，一个用于自主数据准备的大语言模型驱动的智能体系统。DeepPRep 通过与环境进行迭代式、基于执行的交互来构建数据准备流水线；该环境会生成中间表状态并返回运行时反馈。与一次性代码生成和线性 ReAct 风格的交互不同，后两者只能沿单一轨迹进行局部修正，而 DeepPRep 通过基于树的智能体推理来组织流水线构建。当执行反馈表明早先的决策并非最优时，这种设计能够支持结构化探索和非局部修正。具体而言，给定源表和目标模式，DeepPRep 通过包含以下三个步骤的迭代循环运行：

- **规划：**观察当前执行状态，并综合制定高层计划，将现有表转换为目标模式。
- **编排：**将计划实例化为带有指定参数的可执行算子流水线。
- **执行：**执行流水线以生成新的表状态，并返回运行时反馈，供后续决策使用。

通过与执行反馈进行迭代交互，DeepPRep 逐步完善决策，并最终收敛到一个数据准备流水线，该流水线能够生成符合目标模式的表。

示例 1. 请看图2中的示例。目标是构建一张表，利用受众评分分析不同类型影片中导演的表现。遵循 *ReAct* 风格交互的线性智能体可能会过早地连接并聚合数据，从而在类型列中产生*“Sci-Fi, Action”*之类的多值条目，违反类别必须为单值的目标模式约束。要纠正这一错误，需要重新审视电影表中较早做出的清洗决策，即在重新应用下游连接和聚合之前插入 展开 操作。

相比之下，*DeepPRep* 可以自然地支持这一过程。它无需重新启动整个流程，而是可以回溯到做出错误决策时的执行状态，扩展出一条采用正确操作的替代分支，并复用现有部分流水线中有效的算子前缀，从而得到正确结果。

挑战与解决方案。我们研究上述智能体框架带来的研究挑战，并概述相应的解决方案。

首要挑战是在执行反馈的基础上，如何支持结构化探索和非局部回溯。为此，DeepPRep 在流水线构建过程中维护一棵显式的物化执行状态树，其中节点表示已执行算子生成的中间表，边则记录已应用的算子流水线。这种表示方式能够将执行反馈准确归因于先前的决策点，使智能体能够在下游发生失败时回溯并修改上游选择。在此结构基础上，DeepPRep 引入了直接作用于该树的动作结构化大语言模型交互，使智能体能够根据执行反馈逐步扩展、修改或回溯分支。

第二个挑战是如何有效训练大语言模型进行基于树的智能体推理。一种直接的解决方案是采用基于结果级奖励 [14, 21] 的强化学习。然而，这种设置带来了两个难题。第一，奖励稀疏：冗长且相互依赖的流水线使得仅凭结果不足以学习结构化探索和非局部回溯。第二，数据稀缺：现有 ADP 数据集侧重于简单、短小的流水线 [20, 27], ，限制了对多步、具备执行感知能力的推理的监督。为应对这些挑战，我们提出渐进式智能体训练框架：(i) 通过基础算子使用方式和基于树的交互模式初始化模型；(ii) 通过多轮强化学习优化策略，并采用同时考虑结果级和过程级信号的混合奖励；(iii) 利用数据合成构建多样且复杂的 ADP任务用于训练。

<style 编号='1'>贡献。 我们的贡献总结如下。
（1）我们研究了支持广泛数据准备算子的自主数据准备问题（第 2 节），并提出了一种大语言模型驱动的智能体框架（第 3 节）。（ 2 ）我们提出了两种技术来应对这一问题的挑战：用于推理的基于树的智能体推理（第 4 节），以及用于智能体优化的渐进式智能体训练（第 5 节）。（3）我们在真实世界的 ADP 基准上进行了大量实验。结果表明，DeepPRep 的准确率可与闭源模型相媲美（例如，GPT-5），且 15× 推理成本更低；同时，在开源基线中实现了最先进的性能，并能很好地泛化到各种数据集。

（4）我们发布了源代码、数据集以及一套全面的训练模型权重（参数规模从 0.5B 到 14B），使用户能够根据不同的计算预算部署 ADP 智能体[1]。

2 预备知识

2.1 问题形式化

源数据与目标数据。我们考虑一组源表$S = \{S_1、\ldots\ldots、S_n\}$。每个源表表示为 $S_i = (\Sigma_i, D_i)$，其中Σ_i表示 S_i的模式，D_i表示符合Σ_i的一组元组。模式Σ_i涵盖表级和列级规范，定义为 $\Sigma_i = (\tau_i, C_i)$。这里，τ_i表示概述该表语义含义的文本描述，$C_i = \{c_i^1、\ldots\ldots、c_i^{m_i}\}$表示列规范集合。每个列规范$c_i^j$都包含列级元数据，例如列名、数据类型和语义描述。实际应用中，源表的模式Σ_i可能仅部分指定，甚至完全缺失。在这种情况下，可以根据D_i中的原始数据推断表级和列级规范。例如，图2中的每个源表S_i都包含一个模式Σ_i，该模式可能由表级和列级信息指定；此外还包含一组D_i数据元组，例如，(1，“侏罗纪公园”，101，“科幻”)。

类似地，目标表 表示为 $T = (\Sigma^*, D^*)$，其中$\Sigma^* = (\tau^*, C^*)$指定目标模式，D^*表示要生成的目标元组。请注意，在自主数据准备中，只有目标模式 Σ^*作为输入给定，而目标元组D^*是未知的，必须通过转换和清洗源数据来生成。例如，在图2中，目标模式 Σ^*由表级描述（例如，“我需要一张包含……的表”）和一组列规范指定，每个列规范都包含一个列名（例如，“导演_姓名”）以及相应的语义描述（例如，“导演的姓名。”）。

数据准备流水线。我们将数据准备过程建模为对源表应用一系列算子。具体而言，算子表示一种基本的数据转换。形式上，算子 o 由其类型 κ和参数θ确定，并定义一个函数 $o_\theta : \mathbf{T} \to \mathbf{T}$，其中 T 表示表的集合，每张表的形式为 (Σ, D)，如前所述。给定输入表集合 $\mathbf{T}_{\text{in}}$，算子会生成一个输出表集合，即$\mathbf{T}_{\text{out}} = o_\theta(\mathbf{T}_{\text{in}})$。请注意，不同的算子类型涵盖不同类别的数据转换，例如筛选、连接、聚合和清洗。DeepPRep 当前支持的算子类型将在第 2.2 节介绍。

接下来，我们将数据准备流水线（简称流水线）定义为一系列有序算子，这些算子被迭代应用，将一组源表 S 转换为目标表 $\hat{T}$。我们使用符号“→”表示相邻算子的从左到右应用顺序，即，一个算子产生的输出会作为输入传递给其右侧的算子。形式上，流水线 P 定义为在 S 上对 k 算子进行有序组合，即，

$$\mathcal{P} = (\mathcal{S}, o_{\theta_1}^{(1)} \to o_{\theta_2}^{(2)} \to \cdots \to o_{\theta_k}^{(k)}). \tag{1}$$

给定一组源表 S，执行流水线 P 会生成一系列中间表集合，即 $\mathbf{T}_0 = \mathrm{S}$，$\mathbf{T}_i = o_{(i)\theta_i}(\mathbf{T}_{i-1})$ $i = 1、\ldots\ldots、k$，</style}其中，T表示流水线的最终输出，其中仅包含目标表 $\hat{}$，即 $\mathbf{T}_k = \{\hat{T}\}$。以算子 Deduplicate (movies, [id], first) 为例。其算子类型 κ 为 Deduplicate，参数 θ 指的是 (movies, [id], first)。在源表上执行此算子会生成包含已清理电影的中间表，而评分 表和导演 表保持不变。

问题陈述。给定一组源表 $S = \{S_1、\ldots\ldots、S_n\}$、目标模式 Σ^* 以及一组可用算子类型 O，自主数据准备（ADP）问题是寻找一条最优数据准备流水线 P，将 S 转换为目标表 $\hat{T} = (\Sigma^*, \hat{D})$，其中 $\hat{D}$表示由 P 生成且满足目标模式 Σ^*的元组。我们假设存在真值目标表 $T^* = (\Sigma^*, D^*)$，该表仅用于评估。因此，流水线的最优性由 D 相对于 D^* 的质量来定义，并依据任务特定的评估指标进行衡量；相关指标将在实验设置（第 6.1 节）中介绍。

示例 2. 图*2*展示了一个 *ADP*任务示例，其中包含三个源表：电影、评分和导演。目标是生成一个符合给定目标模式的目标表。目标模式既描述了表的整体语义，也规定了其所需结构。具体而言，它对预期内容进行表级概述，并指定列级要求，包括列名（例如，类型）、语义含义（例如，电影类型类别）和数据约束（例如，单值）。为实现这一目标，我们构建了一条由多个算子组成的数据准备流水线，例如去重和连接。每个算子都会作用于当前的表集合，并生成更新后的中间表。例如，应用连接会依据电影标题合并电影表和导演表，生成名为 *movies_directors_join* 的表，其中整合了电影数据与导演姓名，同时保持评分表不变。执行完流水线中的所有算子后，便会生成目标表，如图*2*右侧所示。

2.2 支持的数据准备算子

DeepPRep 支持一系列数据准备算子，并采用可扩展设计，可根据需要纳入新的算子类型。我们共考虑 31 个算子，分为八类，涵盖数据准备流水线中的主要转换模式，包括数据清洗、归一化、结构重塑、表组合和聚合。

2.2.1 数据清洗。数据清洗算子通过检测、修复或删除错误、缺失或重复的记录来提升数据质量，通常应用于数据准备流水线中。

- DropNA(table, subset, how) 会删除指定列中含有缺失值的行，其中 subset 用于指定列，how（“any”或“all”）用于控制删除条件。
- MissingValueImputation(table, column, mode) 使用统计策略填补指定列中的缺失值，其中 mode 用于选择方法（例如，均值、中位数或众数）。
- 去重（table, subset, keep）会根据指定的列移除重复行，并根据 keep 的设置保留首次或最后一次出现的行。
- 错误检测(table, 列, func) 根据用户定义函数识别指定列中的无效记录。

[1]我们将在获得机构批准后公开相关材料。

• 异常值检测(table, 列, 动作)检测指定数值列中的统计异常值，动作决定是移除还是标记这些异常值。

2.2.2 值规范化。 值规范化算子用于统一字段值和数据类型，以确保语义一致性。

• ValueTransform(table, column, func) 对指定列中的值应用用户定义的转换函数，以统一其格式。

• StandardizeDatetime(table, column, format) 将指定列中的值转换为用户定义的日期时间格式。
• CastType(table, column, dtype) 将指定列中的值转换为由 dtype 定义的目标数据类型。

2.2.3 模式编辑。 模式编辑算子用于修改表结构和列语义，实现列选择、列重命名、派生和重构，从而使原始数据与目标模式保持一致。

• RenameColumn(table, rename_map) 根据从原始名称到目标名称的指定映射对列进行重命名。
• AddNewColumn(table, name, func) 使用逐行函数，根据现有数据计算并添加新列。• DropColumn(table, columns) 删除指定的列。

• SplitColumn(table, source, target, func) 使用用户定义的拆分函数将一列拆分为多列。
• Concatenate(table, columns, target, func) 使用格式化函数将多列合并为一列。• SelectColumn(table, columns) 仅保留指定的列。

• Subtitle(table, title, target_col) 添加一列，并填入固定的标题或副标题数值。

2.2.4 行选择。 行选择算子通过筛选、排序或选择行子集来处理表。

• Filter(table, func) 选择满足给定条件的行。
• Sort(table, by, ascending) 根据指定列对表格中的行进行排序，ascending 用于控制排序方向。• TopK(table, k) 保留表格中排名前 k 的行。

2.2.5 聚合。 聚合算子按组汇总数据，通过对记录分组并计算统计数据，将行级数据转换为精简且适用于分析的表示形式。

• GroupBy(table, by, agg) 按指定列对行进行分组，并应用 agg 定义的聚合函数。• Count(table) 返回表格中的总行数。为方便起见，我们将 Count 视为基本聚合算子，但它也可以看作 GroupBy 的一种特殊情况
不使用分组键。
• CalculateStatistic(table, stat, func) 使用用户定义的函数计算表的全局统计量，返回一个用于概括表级属性的标量值。

2.2.6 表组合。 表组合算子将多个表中的数据整合为统一表示，从而支持多来源数据准备流水线。

• Join(left, right, on, how) 根据指定的键列合并两个表，连接类型由 how 定义。
• 并集(表, how)将多个模式兼容的表垂直合并，并由 how 决定是否保留或移除重复行。

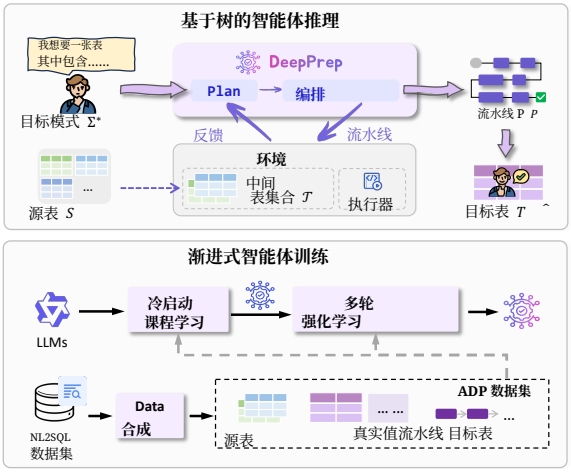


图 3：DeepPRep 概览。

• 追加(表, other)通过垂直拼接行将一个表追加到另一个表，而不像并集那样执行重复项消除。

2.2.7 表重塑。 表重塑算子可重新组织表的结构布局，例如在宽格式和长格式之间转换，或展开嵌套字段。

• Pivot(table, index, columns, values, aggfunc) 通过聚合指定索引和列标识符交叉处的值，将表从长格式重塑为宽格式。

• Stack(table, id_vars, value_vars) 将多个列折叠为键值对，从而将表重塑为长格式。
• WideToLong(table, stubnames, i, j) 通过解析列命名模式，将宽格式数据重塑为长格式。• Transpose(table) 交换表的行和列。

• Explode(table, column) 将包含列表值条目的列展开为多行。

2.2.8 程序合成。程序合成算子提供了一种通用机制，用于处理无法通过上述预定义算子表达的转换。

• ExeCode(tables, target, func) 通过执行由大语言模型合成的 Python 代码，根据输入表生成目标表，从而实现无法通过上述预定义算子表达的转换。

3 DeepPRep 概述

DeepPRep 采用智能体框架，自主编排数据准备流水线，如图 3 所示。给定一组源表 S 和目标模式 Σ^*，DeepPRep 会与 环境 迭代交互，由环境执行数据转换并提供反馈。环境维护由一组中间表表示的执行状态，并以源表 S 初始化该状态。算子实例通过具体的执行后端（例如，Python 解释器）在环境中执行，将转换应用于当前状态，并生成更新后的表以及执行反馈。给定目标模式 Σ^*，DeepPRep 通过一个迭代过程与环境交互，在该过程中交替执行规划、编排和执行 步骤，具体如下所述。

• 在规划 步骤中，智能体观察当前执行状态，并综合生成一份高层文本计划，概述如何将现有表转换为符合目标模式的表。

• 在编排 步骤中，DeepPRep 会将此计划转换为可执行流水线，方法是选择合适的算子类型，并使用特定参数对其进行实例化。

• 在执行 步骤中，编排好的流水线会提交至环境，由环境执行算子、利用新生成的中间表更新其内部状态，并返回执行反馈，例如样例输出或错误跟踪信息。

通过与执行反馈进行迭代交互，DeepPRep 逐步改进规划和编排决策，并最终收敛到能够生成所需目标表的流水线。

示例 3. 考虑图*2*所示的 *ADP*任务。环境使用源表进行初始化。在每次迭代中，*DeepPRep* 会观察当前执行状态，包括序列化的表内容和可用的模式信息。

• 为了使数据与目标模式对齐，智能体首先生成高层文本计划，例如“清理电影表，然后将其与评分表整合”。

• 基于此计划，编排步骤会实例化一条流水线，即 一系列算子，例如 去重(...) 后接值转换(...)。随后，这些编排好的算子会提交至环境执行。

• 执行时，环境会将算子应用于当前状态，并返回执行反馈。具体而言，如果执行成功，环境会更新其内部状态，将清洗后的电影表纳入其中，同时保留原始评分表和导演表。否则，执行错误会被捕获并传递给智能体，智能体则利用这些反馈调整后续的规划和编排决策。

这一迭代过程会持续进行，直到生成的表满足目标模式和质量要求。此时，*DeepPRep* 会终止并输出最终的数据准备流水线。

为支持该框架，我们设计了两个关键组件：基于树的智能体推理和渐进式智能体训练。

基于树的智能体推理。智能体流水线编排面临的关键挑战，是如何在执行反馈的支持下进行结构化探索和回溯。具体而言，流水线构建涉及相互依赖的算子决策，早期选择之后可能会被证明并非最优，甚至不可行。ReAct [2] 和 CoT [37] 等线性智能体推理机制沿着单一决策轨迹进行推理，不会保留备选路径。因此，一旦执行候选流水线，早期错误就可能传递到后续步骤；若没有能够保留历史状态的结构化表示，恢复就会变得困难。

为应对这一挑战，我们引入了一种基于树的智能体推理机制。DeepPRep 不再固守单一的线性执行轨迹，而是将流水线构建表示为一棵搜索树，其中每个节点对应一个关联着一组中间表的规划状态，每条边则表示应用一个候选算子流水线。这种树结构保留了历史状态和备选构建路径，使系统能够根据执行反馈进行系统化探索和回溯。基于这种树表示，我们进一步定义了大语言模型在树上的交互模式和决策逻辑，使智能体能够在流水线构建过程中推理应扩展、修订或放弃哪个分支。我们将在第 4 节详细介绍这种基于树的推理与交互机制。

渐进式智能体训练。关键挑战在于如何有效训练大语言模型，使其具备基于树的智能体推理能力。一种直接的方法是应用现有的强化学习方法，利用结果级奖励优化模型。然而，这种设置面临两大困难。第一，奖励稀疏性：数据准备流水线通常包含较长且相互依赖的算子序列，仅依靠最终结果是否正确，很难让模型学会结构化探索和非局部回溯行为。第二，数据稀缺：现有的 ADP 数据集主要涉及简单转换，算子序列也较短 [20, 27],，无法为学习依赖执行反馈的多步推理提供充足监督。

为解决奖励稀疏性问题，我们提出一种冷启动课程，通过初始化让模型掌握数据准备算子的基础知识以及基于树的智能体推理过程。在此基础上，我们进一步采用多轮强化学习，并引入混合奖励机制，将结果级信号与过程级反馈相结合。为缓解数据稀缺问题，我们开发了一种数据合成方法，以有意义的分析转换为基础生成复杂的 ADP 任务，同时注入多样的数据质量问题，而不破坏预期的转换逻辑。我们将在第 5 节详细介绍智能体训练策略。

4 基于树的智能体推理

为了在复杂依赖和执行反馈下构建数据准备流水线，DeepPRep 采用基于树的智能体推理机制，以支持结构化探索和非局部回溯。具体而言，我们引入智能体推理树（第 4.1 节），用于显式保留历史状态和备选数据准备流水线。在此树结构的基础上，我们通过一组明确的智能体交互动作定义基于树的智能体推理过程（第 4.2 节）。这些动作规定了大语言模型如何观察当前树状态、提出数据准备计划并生成有前景的流水线。

4.1 智能体推理树

我们将智能体推理树定义为一棵有向树 $G = (N, E)$，它表示从初始源表 S 开始构建流水线的结构化搜索空间。

• **节点**（N）。每个节点 $n \in N$对应一个环境状态，该状态由环境维护的当前中间表集合表示。具体而言，根节点 n_0 表示以源表 S 初始化的初始状态。

• **边**（E）。有向边 $e = (n_i \xrightarrow{o} n_j) \in E$ 表示由智能体在状态 n_i下编排并执行一个算子实例 o（即带有特定参数的算子类型）所引发的状态转移。具体而言，执行 o 会将环境状态从 n_i 更新为 n_j，从而生成更新后的中间表。

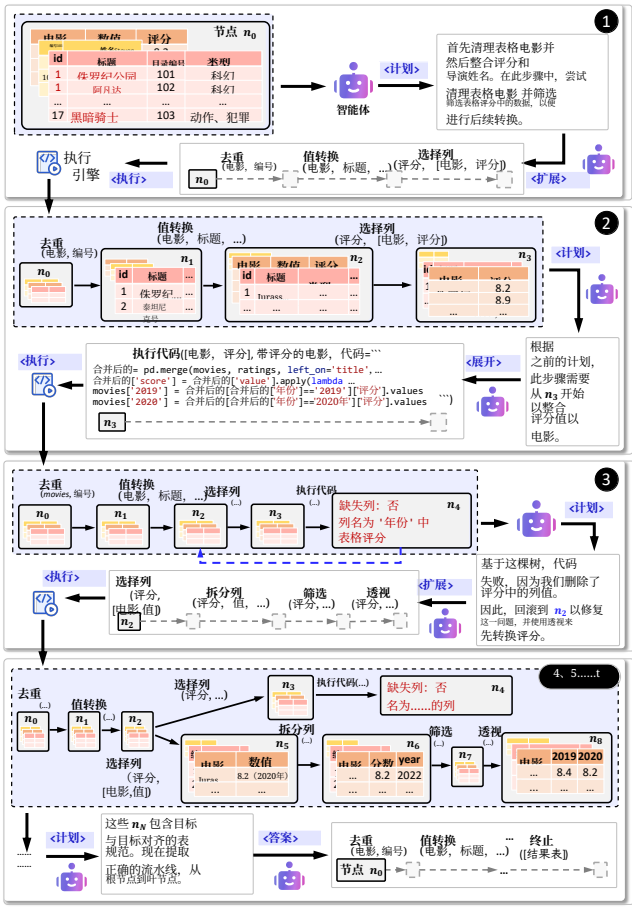


图 4：基于树的智能体推理示例。

在智能体推理树中，每条边都对应一个算子实例，而从一个节点到另一个节点的路径则表示一个算子序列，即一个数据准备流水线。

4.2 基于树的智能体推理

我们提出的基于树的智能体推理，其核心思想是通过结构化的大语言模型交互模式，构建一个迭代式决策过程。具体而言，大语言模型会自主生成智能体交互动作，以操作已实例化的智能体推理树，并根据执行反馈逐步扩展、修订或回溯树的分支。

智能体推理框架。 给定源表 S 和目标模式 Σ^*，该框架会生成一条推理轨迹 R，并最终基于该推理轨迹 R生成答案 A。更形式化地说，我们通过一个概率模型 Pr(·) 对推理过程和基于输入的答案生成进行建模：

$$\Pr(A, R \mid S, \Sigma^*) = \underbrace{\prod_{t=1}^{T} \Pr(R_t \mid R_{<t}, S, \Sigma^*)}_{\text{Reasoning Process}} \cdot \underbrace{\Pr(A \mid R, S, \Sigma^*)}_{\text{Answer Generation}}. \quad (2)$$

其中，T 表示推理过程的长度，R_t 表示智能体在步骤 t 的推理结果。

推理过程通过采用基于大语言模型的智能体，以生成式方式在智能体推理树 G 上进行搜索来实现。具体而言，该过程在计划、扩展和执行步骤之间迭代交替：根据当前树的观测，为当前步骤生成逻辑计划，扩展新的分支，并执行扩展出的算子以更新当前树。基于这一思路，我们将公式（2）中的逐步推理过程分解为以下公式。

$$\Pr(R_t \mid R_{<t}, S, \Sigma^*) = \underbrace{\Pr_{\text{LLM}}(\psi_t \mid \psi_{<t},\ G_{t-1},\ \Sigma^*)}_{\text{计划}} \cdot \underbrace{\Pr_{\text{大语言模型}}(\mathrm{P}_t \mid G_{t-1},\ \psi_t)}_{\text{展开}} \cdot \underbrace{\Pr_{\text{环境}}(G_t \mid G_{t-1},\ \mathrm{P}_t)}_{\text{执行}}. \quad (3)$$

其中，$\Pr_{\text{LLM}}$ 表示大语言模型智能体所诱导的生成分布，ψ_t P_t 表示智能体在步骤 t中生成的逻辑计划和展开后的算子流水线，而 $\Pr_{\text{Env}}$ 表示将 P_t 实例化以获得更新后的智能体推理树 G_t的环境转移分布。

智能体推理的动作。 我们使用与公式 (2) 和公式 (3) 中各组件相对应的控制标签来规范这种交互。

- 计划动作会为下一步推理生成高层逻辑计划。给定当前的智能体推理树、目标模式 Σ^* 和先前的计划，智能体会生成计划 ψ，明确预期的推理决策。该计划会分析当前环境状态，以确定是继续扩展树，还是终止推理过程。如果需要进一步探索，则会确定要扩展的父节点以及要应用的候选操作。• expand。此动作会生成从选定父节点 n_{parent} 扩展出的候选算子流水线。根据计划 ψ，智能体会生成算子序列 P，以扩展节点 n_{parent}。为确保节点选择一致，父节点需通过前缀匹配约束指定：智能体必须引用从根节点到该节点的完整算子序列。此约束可避免含糊或基于索引的节点引用，并确保所提议的扩展与记录的转换历史保持一致。• execute。此动作通过环境应用候选算子流水线并执行状态转移。具体而言，给定候选流水线 P_t，环境会在所选父节点处的物化表上依次执行其中的算子。每成功应用一个算子，就会创建一个新的中间状态节点来存储生成的表，并添加一条从前一个状态节点指向新节点的有向边。对 P_t 中的所有算子重复此过程后，会生成一串新节点，并将 G_{t-1} 更新为 G_t。如果执行因运行时异常而失败（例如，列不匹配或类型错误），则不会创建新的状态节点；相反，父节点会被标注为失败状态，并记录错误跟踪。执行反馈将指导后续动作。• answer。此动作会终止推理过程并返回最终流水线。当某个叶节点状态满足目标模式 Σ^* 时，就会触发此动作。与该节点关联的根到叶算子路径将被提取为最终流水线 P^*，并作为输出返回。

智能体推理过程。 我们将上述定义的动作标签加入大语言模型的词汇表，使基于树的推理能够表示为结构化语言建模过程。

在推理过程中，模型会迭代生成与已定义交互动作相对应的带标签输出。具体而言，在每次迭代中，模型首先以plan.../plan的形式生成计划，用于指定下一步推理决策。基于该计划，模型会预测一个由expand.../expand包裹的算子扩展。随后，生成的流水线由环境执行，并通过execute.../execute记录所得状态和执行反馈。上述过程会持续迭代，直至某个状态满足目标模式。当满足该条件时，使用answer.../answer返回最终流水线，从而终止推理过程。

示例 *4*。图 *4* 展示了图*2*所示任务的一个推理轨迹示例。 ① 初始化：推理从节点 n_0开始，该节点对应于一个与源表 S 关联的初始状态。在推理过程中，模型首先生成一个〈*plan*〉动作，指定一个逻辑计划，用于清理电影表并在评分表中选择相关列。随后，〈*expand*〉动作生成去重、值转换和选择列算子，并通过〈*execute*〉应用这些算子，生成新状态，即 n_1、 n_2 和 n_3。 ② 执行失败：后续的〈*plan*〉和〈*expand*〉动作提出使用 *ExeCode* 算子整合评分数据。然而，在〈*execute*〉期间引发了缺少列的运行时错误，该转换在 n_4 处被记录为失败，且未创建新的有效状态。 ③ 回溯与替代扩展：基于记录的失败反馈，新的〈*plan*〉分析该错误，并尝试通过在应用选择列算子时保留评分表的列值来调试问题。具体而言，它选择 n_2 作为调试和进一步探索的父状态。除了应用更新后的选择列算子外，〈*expand*〉动作还提出了一个替代算子序列（拆分列、筛选、透视）。这些算子被应用以生成新状态（n_5、 n_6、 n_7、 n_8）。t ◯ 终止：当叶状态满足目标模式 Σ^*时，将发出〈*answer*〉动作，以提取相应的根到叶算子路径作为最终流水线。

5 智能体训练框架

我们提出了一种用于优化的渐进式智能体训练框架。为缓解奖励稀疏性，该框架首先通过冷启动课程学习进行初始化（第 5.1 节），随后进入多轮强化学习阶段，并采用混合奖励机制（第 5.2 节）。为缓解数据稀缺问题，我们开发了一种数据合成方法（第 5.3 节），用于生成多样化的 ADP 任务，以实现有效训练。

5.1 冷启动课程

为缓解未经训练的大语言模型无法生成可执行流水线这一“冷启动”问题，我们设计了冷启动课程模块，以初始化大语言模型针对结构化且由执行驱动的规划所需的推理能力。具体而言，预训练大语言模型通常缺乏对数据准备算子语法和前文所述基于树的智能体推理过程的明确理解。因此，若没有专门的监督，模型就无法为ADP任务生成有意义的动作序列。为解决这一问题，我们采用两阶段监督微调（SFT）课程学习。

阶段 1：算子语法学习。此阶段聚焦于各个算子的语法和语义。给定源表和如公式（1）所定义的真实算子流水线 $\mathrm{P}_{\text{total}}$，该流水线由一系列算子组成（$o_{(1)}\ldots,o_{(N)}$）。我们首先依次执行这些算子，收集生成的中间表集合序列 $\{\mathrm{T}_0,\ldots,\mathrm{T}_N\}$。然后，我们采样一段连续的算子子序列 $\mathrm{P}_{\text{sub}}=(o_{(i)}\ldots,o_{(j)})$，并获取对应的输入中间表集合 $\mathrm{T}_{\text{in}}=\mathrm{T}_{i-1}$ 和输出中间表集合 $\mathrm{T}_{\text{out}}=\mathrm{T}_j$。通过这种方式，我们构建算子语法学习数据集 D_{op}，其中包含训练三元组 $(\mathrm{P}_{\text{sub}},\mathrm{T}_{\text{in}},\mathrm{T}_{\text{out}})$。令 $\phi(\cdot)$ 表示序列化函数。该函数通过拼接所有序列化后的算子来序列化 P_{sub}，并将中间表集合 T 序列化为 Markdown 格式。我们采用监督微调来优化以下损失 L_{op}:

$$\mathcal{L}_{\text{op}}=-\sum_{(\mathcal{P}_{\text{sub}},\mathcal{T}_{\text{in}},\mathcal{T}_{\text{out}})\in\mathcal{D}_{\text{op}}}\log \Pr_{\text{LLM}}(\phi(\mathcal{P}_{\text{sub}})\mid\phi(\mathcal{T}_{\text{in}}),\phi(\mathcal{T}_{\text{out}})). \tag{4}$$

通过公式（4），智能体学会识别算子函数签名和参数格式，同时将算子执行逻辑与更高层次的规划决策相分离。

阶段 2：推理过程学习。 该阶段使模型与智能体树式推理 机制（第 4.2 节）保持一致 .

为此，我们通过从强大的教师模型（例如，DeepSeek-R1）中提炼知识，合成一个高质量推理轨迹数据集。具体而言，对于由源表 S 和目标模式 Σ^* 定义的每个 ADP任务，我们旨在生成一条真实轨迹 τ。为增强鲁棒性，我们施加输入扰动 Φ，例如随机打乱源表中的行和列。随后，我们采用上下文学习 [8] 来引导教师模型。通过使用详细的提示词来示例公式（2）中定义的基于树的推理过程，我们生成一组候选轨迹 U。为统一训练目标，我们将每条轨迹表示为一个序列 $\tau=(r_1,\ldots\ldots,r_m)$。其中，初始子序列（$r_1,\ldots\ldots,r_{m-1}$）对应于公式（3）中定义的迭代推理步骤（$R_1,\ldots\ldots,R_T$），而最终元素 r_m对应于公式（2）中的答案生成 A。这些候选轨迹通过过程奖励函数 $R_{\text{llm}}(\tau)$（详见第 5.2 节）进行筛选，仅保留排名最高的轨迹，以形成最终数据集 U^*。最后，在 U^* 上对模型进行微调，以最小化掩码负对数似然:

$$\mathcal{L}_{\text{reason}}=-\sum_{\tau\in\mathcal{U}^*}\sum_{t=1}^{m}M_t\cdot\log \Pr_{\text{LLM}}(\phi(r_t)\mid\phi(r_{<t}),\phi(\mathcal{S}),\Sigma^*). \tag{5}$$

在此， r_t封装了当前树状态和智能体的生成响应。序列化函数 $\phi(\cdot)$将这些结构化组件转换为线性词元序列。关键的是，我们引入了损失掩码 M_t，以确保仅针对与智能体输出相对应的词元计算梯度。

5.2 使用HybridReward的多轮强化学习

动机。 数据准备流水线生成是一个带有执行反馈的序贯决策过程，其中每个动作都会影响后续状态以及未来的算子选择。优化

此类行为无法简化为单轮生成。相反，策略应在多轮交互轨迹上进行训练。在每一轮中，智能体生成结构化动作（例如，plan和expand），环境执行所提出的算子并返回反馈。随后，后续决策以这些执行结果为条件。

基本思路。上述设定促使我们开发一种多轮强化学习方法，在该方法中，策略针对动作序列而非单次输出进行优化。具体而言，在我们的交互框架下，每条轨迹都包含 (i) 智能体生成的词元（动作及其内容）和 (ii) 环境生成的词元，例如序列化的中间表和执行轨迹。特别是，由于环境生成的词元是确定性的执行输出，而非策略决策，因此不纳入策略梯度优化。也就是说，我们使用二进制词元掩码，移除 execute 块内词元的梯度，并仅针对 plan、expand 和 answer 中的智能体决策词元优化策略。

我们使用组相对策略优化 (GRPO) [34], 来优化策略。该方法从组级奖励统计数据中估计基线，而不是单独训练价值网络，从而支持对多轮轨迹进行更稳定的优化。然而，仅将最终流水线正确性作为奖励会导致学习信号稀疏且延迟。因此，我们提出一种混合奖励设计，详述如下。

混合奖励。 我们设计了一种混合奖励机制，通过评估结果质量和推理行为，提供更密集的学习信号。轨迹 τ的总体奖励 $R(\tau)$ 定义为以下三个组成部分的加权和：

$$R(\tau) = \alpha \cdot R_{\text{out}}(\tau) + \beta \cdot R_{\text{part}}(\tau) + \gamma \cdot R_{\text{llm}}(\tau). \tag{6}$$

其中， R_{out}、 R_{part} 和 R_{llm} 分别对应严格正确性、部分进展和推理过程信号，而 α, β, γ 是用于平衡各自贡献的超参数。下面分别详细介绍这三个组成部分 R_{out}、 R_{part} 和 R_{llm}。

结果奖励 R_{out}。 该奖励根据最终输出正确性提供直接的成功信号。当发出 答案 动作时，会对生成的目标表 T 进行评估：

$$R_{\text{out}}(\tau) = \mathbb{I}(\hat{T} \equiv T^*). \tag{7}$$

其中， $\mathbb{I}(\cdot)$ 表示指示函数。

部分奖励 R_{part}。 当未达到严格正确性时（$R_{\text{out}}(\tau) = 0$），该奖励提供部分进展信号。我们衡量生成表 T 与真实值T^*之间的相似度，并定义 $R_{\text{part}}(\tau) = \text{avg}(S_{sch}, S_{shp}, S_{cnt})$，其中 $S_{sch}, S_{shp}, S_{cnt}$分别评估模式、形状和内容的对齐情况：

$$\begin{aligned} S_{sch} &= \frac{|C_{\hat{T}} \cap C_{T^*}|}{|C_{\hat{T}} \cup C_{T^*}|}, \quad S_{shp} = \exp\left(-\left|\frac{|\hat{D}| - |D^*|}{|D^*|}\right|\right), \\ S_{cnt} &= \frac{1}{|C_m|} \sum_{c \in C_m} \frac{\sum_i \mathbb{I}(\hat{D}[c]_i = D^*[c]_i)}{\max(|\hat{D}|, |D^*|)}. \end{aligned} \tag{8}$$

这里， $C_{\hat{T}}$和 C_{T^*} 表示列名集合， $|\hat{D}|$和 $|D^*|$表示行数，而 $C_m = C_{\hat{T}} \cap C_{T^*}$ 是用于逐单元格值比较的匹配列集合。

过程奖励 R_{llm}。 仅依赖数据相似度奖励可能导致奖励投机，这一现象已在其他强化学习场景中得到研究 [36, 44], ，也可能出现在 ADP 场景中（例如，只重命名列而不执行所需的数据清洗）。为抑制这类“走捷径”策略并促进语义一致的决策，我们引入过程奖励来评估推理轨迹的质量。我们使用指令调优大语言模型（例如，GPT-4o）作为评判者，根据第 4 节定义的动作对每条轨迹进行评分。

具体而言，评审器会评估三项标准：(1) 计划—行动一致性，衡量生成的流水线（来自 expand）是否正确实现了计划（来自 plan）。(2) 反馈响应性，检查后续的 plan 输出是否明确回应了先前的执行反馈（例如，报告的错误跟踪信息）。(3) 回溯正当性，验证父节点切换是否有当前分支记录的失败证据作为依据。

5.3 用于训练的数据合成

为了训练我们的智能体模型，我们需要以可执行的数据准备流水线及其来源表和目标表的形式提供监督。由于这类 ADP 任务实例无法大规模获得，我们通过合成来构建这些实例：将 SQL 基准测试转换为由源表、目标表和相应转换流水线组成的 ADP 任务。合成此类训练任务面临两项关键挑战。首先，生成的流水线应对应有意义的分析转换，而非随意拼接的算子链，从而使监督能够反映真实的算子依赖关系。其次，合成的源表应包含多样的数据质量问题，同时保留预期的转换逻辑。过度注入噪声可能会破坏关键字段或关系， [30], 导致难以构建任何能够生成目标表的可行流水线。

基于NL2SQL的ADP任务构建。 为获得真实的自主数据准备任务，我们从NL2SQL基准测试 [26, 42],中实例化这些任务，这些基准测试提供了与分析型SQL查询配对的真实数据库。对于每个真实查询 q，我们执行该查询以生成目标表 T^*，并使用大语言模型生成相应的目标模式规范。我们还将 q 转换为算子流水线：利用大语言模型生成候选流水线，并选择通过执行能够精确复现 T^* 的最短流水线。由此得到一个干净的任务实例，包含源表、目标模式规范、目标表和任务流水线。

可逆噪声注入。 由于 NL2SQL 数据通常是干净的，我们引入可控噪声，以增加训练数据的多样性。我们通过对数据清洗算子应用逆变换来注入噪声，使每种损坏都对应一个有效的清洗步骤。具体而言，我们采样一个算子，并使用大语言模型生成其逆变换逻辑。例如，日期标准化算子的逆变换会将标准化日期（例如，“2023-01-01”）转换为异构格式（例如，“01/01/23”）。每次损坏后，我们都会应用相应的清洗算子，并检查是否恢复了先前的表状态，以验证其可执行性。我们只保留可逆的损坏。重复这一过程，就能生成脏源表及其对应的清洗流水线。最终的真实值流水线由清洗流水线与任务流水线串联而成。

表 1：数据集的统计数据。

数据集	统计数据		流水线长度	# 操作类型
	# 训练集	# 测试集		
Synth-Spider	6,788	2,008	1~28	31
Synth-Bird	-	1,150	2~25	31
Parrot [12]	13,965	1,365	1~8	17

6 实验

6.1 实验设置

数据集。 我们在三个数据集上评估 DeepPRep，其中包括一个公开的 ADP 基准和两个使用我们的合成方法生成的数据集。数据集统计数据汇总于表 1。

(1) **Parrot** [12] 是一个用于将自然语言转换为数据准备流水线的基准。然而，它不提供诸如目标模式描述之类的 ADP 风格任务规范。为使其适用于 ADP 评估，我们使用大语言模型 [33]，根据目标表和算子序列生成相应的任务规范，以扩充每个案例。最终数据集包含 15,330 个案例，流水线长度介于 1 到 8 之间。

需要注意的是，公开基准 **Parrot** [12] 主要包含较短的算子流水线，且算子类型覆盖范围有限。因此，我们使用第 5.3 节中的方法合成两个数据集，以提供更复杂的流水线和更广泛的算子覆盖范围。

(2) **Synth-Spider** 是我们使用 Spider NL2SQL 基准 [42]通过所提方法合成的数据集。利用第 5.3 节中介绍的 ADP 任务合成流水线，我们将 NL2SQL 案例转换为带有可执行算子流水线的 ADP 任务。最终数据集包含 6,788 个训练案例和 2,008 个测试案例，真实流水线长度介于 1 至 28 之间。

(3) **Synth-Bird**是另一个合成数据集，基于 Bird 基准 [26]，构建，该基准包含规模更大的数据库和更复杂的分析查询。我们采用相同的合成流程，从 Bird 验证集生成 ADP 任务，共得到 1,150 个案例，流水线长度介于 2 至 25 之间。由于 Synth-Bird 仅用于评估，因此该数据集只有测试集。与 Synth-Spider 相比，该数据集涉及更长的转换链和更繁重的数据处理。

评估方法。 为进行全面评估，所有模型仅在**Synth-Spider** 训练集上训练，并在两种测试设置下评估：域内 和域外。

（*1*）域内测试集。 我们在**Synth-Spider** 测试集上进行评估。该测试集采用与训练数据相同的流程合成，但使用了互不重叠的查询和数据库。

（*2*）域外测试集。 我们在两个来源和任务格式各不相同的数据集上进行评估。**Synth-Bird** 与训练分布更接近，因为它采用相同的合成流水线生成，但来自不同的基准。**Parrot** 则是一个独立数据集，其任务特征有所不同。

基线。 我们将 DeepPRep 与提示基线 和训练基线进行比较。提示基线使用现成的大语言模型及各种推理策略，而训练基线则通过监督学习或强化学习进行参数更新。

(1) 提示基线。 这些方法使用冻结的大语言模型，采用不同的规划和搜索策略来生成处理流水线。

• CodeGen [28, 32] 提示大语言模型根据源表和目标模式生成数据准备代码。

• Plan-and-Solve（PaS） [37] 采用两阶段策略提示大语言模型：首先生成高层计划，然后生成完整的算子序列，以获得目标表。

• ReAct [2] 是一种线性推理智能体，会根据先前的执行结果迭代预测下一个算子。

• MCTS-OP 使用蒙特卡洛树搜索，通过局部节点扩展和标量奖励来引导搜索。我们遵循 [22] 的框架，并保留其实现中的参数。

• MontePrep [11]使用蒙特卡洛树搜索，并针对模式映射、算子发现、代码合成和细化等动作进行树搜索，而非针对单个算子进行搜索。我们使用作者发布的代码及默认设置。

• DeepAnalyze [43] 是一种智能体式大语言模型，通过监督学习和强化学习生成数据处理程序。

• AutoPrep [20] 是一个用于表格重构和列转换的多智能体框架。它会生成高层计划，然后通过专门的智能体生成数据准备程序。由于其涵盖的算子不如我们的设置全面，我们遵循其原始设计对其进行扩展。

(2) 训练基线。 这些基线使用与 DeepPRep 相同的骨干模型，但采用不同的训练策略，以便我们评估智能体训练框架的效果。

• 模仿学习（IL） [24] 在从强教师模型蒸馏出的高质量推理轨迹上对模型进行微调（例如，DeepSeek-R1）。与针对（输入，输出）对进行标准监督微调不同，这一基线方法学习逐步推理过程。

• RL-GRPO [34] 采用群组相对策略优化进行强化学习，仅使用结果级奖励，不采用我们的课程学习和混合奖励设计。

评估指标。 我们使用三项指标评估所有方法。（*1*）精确匹配准确率（准确率）衡量生成的表格 $\hat{T}$与真实值表 T^*匹配的测试案例所占比例。我们采用 [12], 中的表匹配方法，该方法不受行列置换影响，但要求单元格值完全相等。（*2*）完成率（复杂度）衡量生成最终输出表格的测试案例所占比例。如果执行超过最~~大交互次数限制、触发运~~行时错误或生成空结果，则该案例计为未完成。

(3) 推理成本（成本）衡量每个案例的平均货币成本。对于闭源大语言模型，我们在启用缓存的情况下，按官方定价计算 API成本。对于开源模型，我们测量并行请求下每个案例的端到端运行时间，并根据 GPU实例的小时价格将其换算为成本。具体而言，案例 i 的成本计算方式为 $c_i = p_{\text{gpu}} \cdot t_i/3600$，并报告所有案例的平均值。我们使用一块价格为 0.91 美元/小时的 A800-80G GPU（ https://cloud.vast.ai/）。

大语言模型骨干与 DeepPRep 变体。我们在闭源和开源大语言模型骨干上均实现了 DeepPRep。

（*1*）闭源骨干。我们在最强大的专有模型上直接实现了 DeepPRep，并采用所提出的基于树的智能体推理，包括 GPT-5（即 gpt-5-mini-2025-08-07）和 Claude-4（即 claude-sonnet-4-20250514）。

表 2：DeepPRep 与基线方法在开源大语言模型上的比较。最佳和次佳结果分别以粗体和下划线标出。

(a) Qwen3-14B 上的结果。

方法	Synth-Spider		Synth-Bird		Parrot	
	Acc.	复杂度	Acc.	复杂度	Acc.	复杂度
提示基线						
PaS	7.67	20.67%	3.09	21.86%	9.56	22.36%
MCTS-OP	21.91	72.56%	7.22	78.69%	24.95	83.02%
MontePrep	9.46	19.62%	18.63	38.15%	26.94	57.87%
AutoPrep	36.75	71.61%	10.41	39.89%	28.54	74.05%
CodeGen	45.47	68.18%	29.48	49.20%	30.83	81.08%
ReAct	40.39	69.02%	16.04	46.56%	30.80	76.44%
训练方法（仅在 *Synth-Spider* 上训练）						
IL	61.70	85.26%	48.01	81.23%	37.95	89.39%
RL-GRPO	51.22	77.68%	32.25	67.22%	35.06	82.9%
DeepPRep（我们的方法）	**67.18**	**97.21%**	**54.09**	**92.26%**	**40.44**	**96.41%**

（b）Qwen3-8B 上的结果。

方法	Synth-Spider		Synth-Bird		Parrot	
	Acc.	复杂度	Acc.	复杂度	Acc.	复杂度
提示基线						
PaS	4.68	7.42%	0.65	4.33%	3.09	8.32%
MCTS-OP	12.90	15.19%	0.90	73.56%	17.33	23.71%
MontePrep	8.81	19.97%	13.70	21.96%	25.95	56.08%
AutoPrep	26.44	50.85%	8.52	42.28%	25.20	70.42%
CodeGen	6.27	12.10%	7.47	15.69%	7.27	25.50%
ReAct	29.63	52.59%	10.41	46.22%	22.11	57.97%
DeepAnalyze	35.66	82.02%	1.74	62.17%	29.01	66.67%
训练方法（仅在 *Synth-Spider* 上训练）						
IL	59.65	90.54%	43.30	80.43%	36.15	91.08%
RL-GRPO	47.81	76.54%	29.95	68.69%	32.82	85.05%
DeepPRep（我们的方法）	**65.99**	**97.46%**	**53.39**	**92.78%**	**39.93**	**98.46%**

（2）开源骨干模型。我们基于开源大语言模型构建 DeepPRep，即，Qwen2.5 [18]（参数量分别为 0.5B、1.5B、3B、7B 和 14B）以及 Qwen3 [38]（参数量分别为 0.6B、1.7B、4B、8B 和 14B）。对于这些模型，我们采用智能体训练流程进行训练，然后使用相同的推理框架对其进行评估。

实验设置。对于所有提示方法，温度均设为 0.01。DeepPRep 的最大探索轮次设为 5。对于所有训练方法，我们使用 AdamW优化器，SFT 的学习率为 6×10^{-5}，RL 的学习率为 1×10^{-6}。所有实验均在 16 块 NVIDIA A800-80G GPU 上进行。详细的提示词和超参数配置见 [1]。

6.2 总体比较

实验 1：准确率与完成率比较。 我们在准确率和完成率方面，将 DeepPRep 与提示基线和训练基线进行比较。表 2 报告了 Qwen3-14B 和 Qwen3-8B 上的结果；这两种模型规模的开源骨干模型实力强劲，且常用于代码与数据任务。更多骨干模型的结果将在后续实验中展示。

（*1*）域内测试集。DeepPRep 在 Synth-Spider 测试集上使用两种骨干模型时，均取得了最高的准确率和完成率。使用 Qwen3-14B 时，其准确率达到 67.18（表 2a），比最强的提示基线 CodeGen 高 21.71 个百分点，比最强的训练基线高 5.48 个百分点，完成率也有类似提升。相比之下，通用数据智能体 Deep-Analyze 在同一数据集上的准确率为 35.66。这些提升源于树结构探索与以执行为依据的动作相结合，使智能体能够修正先前的决策，并在流水线构建过程中避免无效的算子序列。

(2) 域外测试。 在单源训练方法下，DeepPRep 在域外数据集上保持明显优势。在 Synth-Bird 上，DeepPRep（Qwen3-14B）的准确率为 54.09，高于 IL 的 48.01；8B 模型的准确率为 53.39，高于 43.30。在 Parrot 上，各方法的排序也呈现出类似趋势。DeepAnalyze 在 Synth-Bird 上的表现低得多（准确率为 1.74），表明当流水线长度和算子依赖关系增加时，其鲁棒性有限。这些结果与我们的训练设计相符：渐进式课程和混合奖励让模型接触多步、由反馈驱动的推理轨迹，而非单轮程序生成。

发现 1：DeepPRep 在所比较的方法中实现了最高的准确率和完成率，并且在不同数据集上表现出良好的泛化能力。

实验 2：成本–准确率权衡比较。我们通过联合评估准确率和成本，在成本–准确率权衡下比较各方法。图 5 展示了不同大语言模型骨干上的提示基线和 DeepPRep 变体，并呈现了两条帕累托前沿：先前帕累托前沿，仅根据基线计算得出；以及我们的帕累托前沿，在加入 DeepPRep 后计算得出。

首先，在大多数设置中，DeepPRep 位于帕累托前沿上或其附近，并在全部三个数据集上拓展了前沿。图 5 中的我们的帕累托前沿曲线相较于先前帕累托前沿向左上方移动，这意味着 DeepPRep 能以相同成本实现更高准确率，或以更低成本实现相近准确率。这一现象出现在 Synth-Spider、Synth-Bird 和 Parrot 上，表明其成本–准确率优势在不同数据集和大语言模型骨干上均保持一致。

其次，DeepPRep 以显著更低的成本达到了接近强大闭源基线的准确率。在 Synth-Spider 上，DeepPRep（Qwen3-14B）以极低成本取得了 67.18 的准确率，其准确率与 Claude-4 上的 ReAct（69.92 的准确率）和 GPT-5（67.03 的准确率）相当，同时成本分别降低了约 57× 和 15×。这得益于我们的设计：使用结构化树式推理，引导规模更小或开源的模型以更少的失败执行次数生成有效的处理流水线。

发现 2：DeepPRep 扩展了成本–准确率帕累托前沿，以显著更低的推理成本达到了接近强大闭源基线的准确率。

6.3 消融研究

实验-3：消融研究。我们评估了智能体训练框架中特定组件的贡献。结果详见表3。

表3a表明，冷启动课程学习很有用。移除此组件会导致所有数据集的准确率和完成率同时下降。例如，

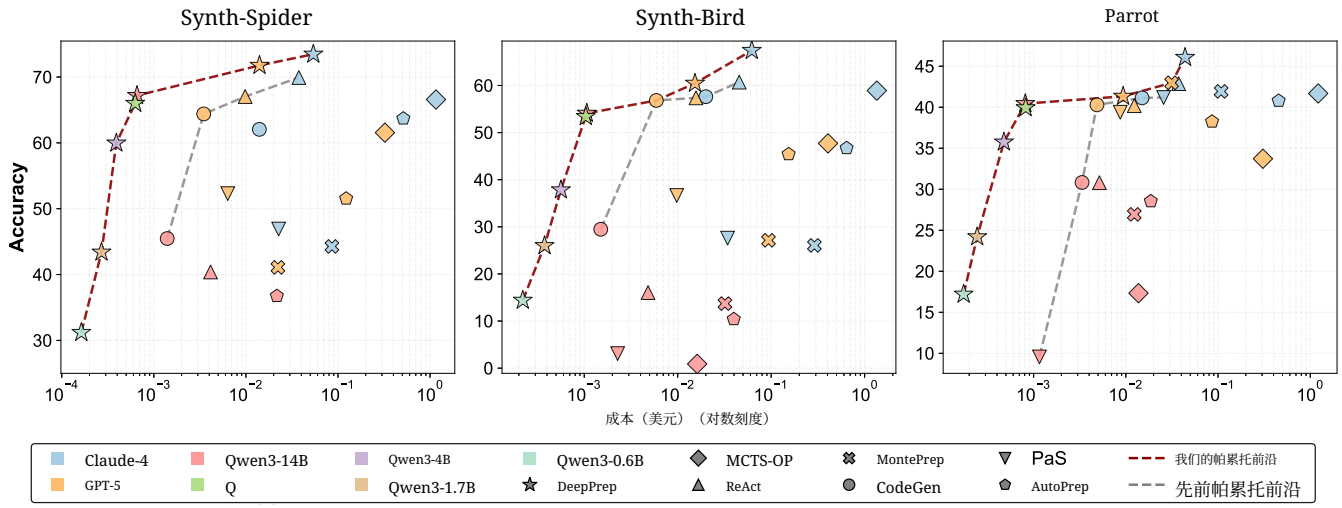


图5：Synth-Spider、Synth-Bird 和 Parrot 数据集上的成本—准确率权衡比较。颜色表示不同的大语言模型骨干，形状表示不同的方法。红色和灰色虚线分别表示我们的方法和先前基线的帕累托前沿。

表3：消融研究（Qwen3-8B）。

(a) 冷启动阶段的消融研究。OSL代表第5.1节中的算子语法学习模块

方法	Synth-Spider		Synth-Bird		Parrot	
	Acc.	复杂度	Acc.	复杂度	Acc.	复杂度
冷启动	**60.66**	**91.63%**	**45.27**	**81.39%**	**36.22**	**92.16%**
不含 OSL	59.65	90.54%	43.30	80.43%	36.15	91.08%

(b) 多轮强化学习阶段奖励的消融研究。

方法	Synth-Spider		Synth-Bird		Parrot	
	Acc.	复杂度	Acc.	复杂度	Acc.	复杂度
DeepPRep	**65.99**	97.46%	**53.39**	92.78%	**39.93**	98.46%
不含 R_{llm}	64.54	96.96%	51.67	92.31%	39.12	94.21%
不含 R_{llm}, R_{part}	61.85	**98.71%**	47.15	**95.57%**	37.22	**99.34%**

Synth-Spider 上的完成率从 91.63 降至 90.54。这种性能下降是因为该组件使模型熟悉算子的语法和参数约束。缺少这种对齐后，模型倾向于生成不可执行的动作，因此无法完成推理轨迹。

表3b 验证了我们在强化学习中奖励函数的设计。首先，移除过程奖励（R_{llm}）会导致准确率下降，这表明 R_{llm} 可以防止“奖励投机”。如果没有语义验证，策略会学着满足表面的模式约束，例如重命名列，却忽略实际的数据转换逻辑。其次，仅依赖结果奖励（不含$R_{\text{out}}, R_{\text{part}}$）会带来最高的完成率，但准确率最低。这表明稀疏奖励会诱导出保守行为。智能体会避免复杂的探索或回溯，以尽量降低运行时错误的风险。因此，它会优先选择安全但不正确的路径，以确保回合成功终止。

发现3：渐进式智能体训练框架的各个关键组件均有助于提升性能，

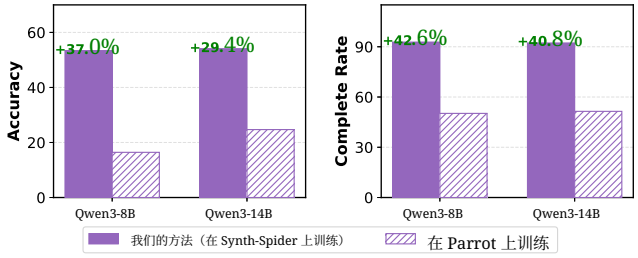


图6：在 Synth-Spider（训练集）和 Parrot（训练集）上训练的模型在 Synth-Bird（开发集）上的评估结果。

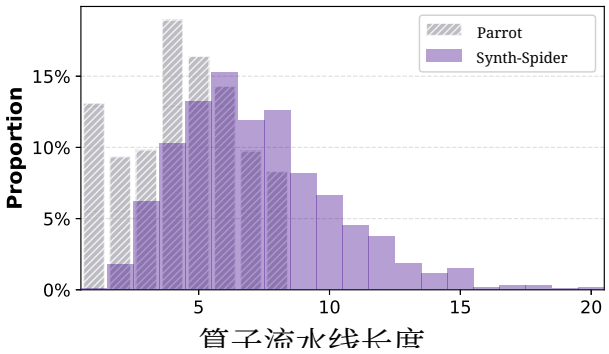


图7：算子流水线长度分布。

消融实验表明，移除冷启动或混合奖励阶段后，性能均会稳定下降。

6.4 训练数据合成的有效性

实验4：合成训练数据的有效性。我们评估在合成的 Synth-Spider 训练集上训练的模型，是否比在公开的 Parrot 训练集上训练的模型具有更好的泛化能力。两种模型均在 Synth-Bird开发集上进行评估。

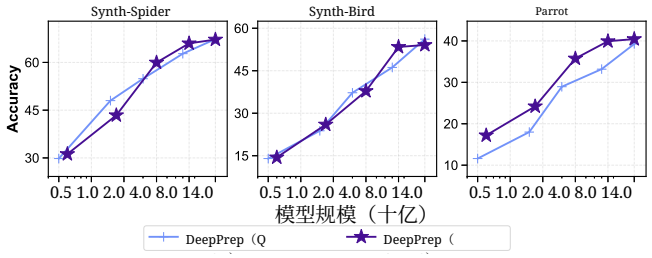


图 8：随着模型规模增大，性能也随之提升。

如图6所示，在 Synth-Spider 上训练可显著提升性能：与 Parrot 相比，Qwen3-8B 和 Qwen3-14B 的准确率分别提高了 37.0 和 29.4 个百分点。完成率的提升更为显著，分别提高了 42.6 和 40.8 个百分点。为了解 Synth-Spider 为何能带来更强的泛化能力，我们在图7中比较了两个训练数据集的算子流水线长度分布。Parrot 的样本集中在较短的流水线上，而 Synth-Spider 则持续包含更多中等长度和较长的算子流水线。这增加了程序复杂性，促进了智能体训练期间基于树的智能体推理，进而提升了模型对未见数据的泛化能力。

发现 4：我们的合成训练数据为智能体训练提供了有效监督，显著提升了未见任务上的准确率和完成率。

6.5 模型规模扩展研究

实验 5：模型扩展研究。我们研究了 DeepPRep 在不同骨干模型规模下的表现。我们使用 Qwen2.5 和 Qwen3 系列中参数量从 0.5B 到 14B 的大语言模型骨干来训练 DeepPRep。图 8 显示，准确率随着模型规模的增大而稳步提升。在 Synth-Spider 上，将 Qwen3 从 0.6B 扩展到 14B，准确率从 31.20 提升至 67.18。在规模相近的模型之间，Qwen3 系列在域外表现上也优于 Qwen2.5。例如，在 Synth-Bird 上，Qwen3-8B 的准确率为 53.39，而 Qwen2.5-7B 为 46.17。这些结果表明，DeepPRep 能够从更强的骨干模型中获益，并保持正向的扩展趋势。

发现 5：随着骨干模型规模增大，DeepPRep 的性能有所提升，这表明 DeepPRep 能够从更强的骨干模型中获益，并保持正向的扩展趋势。

7 相关工作

数据准备。 以往的数据准备工作主要聚焦于示例编程（PBE）或专门的子任务。PBE 系统 [3, 13, 16, 19, 35] 根据输入-输出示例合成转换，适用于规格明确的转换，但每项任务都需要示例级监督。其他系统则针对特定的数据准备子问题 [5, 9, 10, 17, 20, 27, 39] [27] 。例如，Auto-Tables 专注于表重建操作，如透视和堆叠。这类系统侧重于结构变换，但并非为同时处理值级清洗 [4] 和多表集成 [45]而设计。Text-to-Pipeline [12]将自然语言描述映射为处理流水线，但假设输入包含详细的逐步规格说明。相比之下，DeepPRep 面向从高层目标模式描述到端到端的自主数据准备，并支持范围更广的算子空间，涵盖清洗、重塑和集成。

大语言模型驱动的数据准备。 近期研究探索了使用大语言模型构建数据准备流水线。基于提示的方法直接根据自然语言规范生成转换代码 [20, 28, 32]。为纳入执行反馈，ReAct 风格方法将推理与工具操作交替进行，并根据观察到的结果更新后续决策 [2, 41]。此类方法遵循线性交互轨迹，动作按顺序追加。近期研究进一步探索了在推理空间或算子空间中进行树状搜索，包括思维树（ToT） [40] 和基于蒙特卡洛树搜索的智能体 [11, 22]。例如，蒙特卡洛树搜索风格方法通常依靠基于推演的价值估计和标量奖励信号来指导节点选择与反向传播 [15]。然而，数据准备流水线会产生结构化执行反馈，例如模式不匹配、算子执行错误以及中间表状态。DeepPrep 旨在直接基于此类执行状态开展工作，其中每个节点对应物化表，分支决策则由结构化执行反馈而非标量推演统计数据引导。这种设计支持在流水线构建过程中进行算子级修订和状态感知回溯。

面向数据准备的数据合成。 用于数据准备的训练数据有限，尤其是对于需要多步数据准备流水线的任务。NL2SQL 合成方法 [25] 生成查询–结果对，但通常不提供 ADP 训练所需的算子级转换序列。Auto-Tables [27] 提出了一种基于规则的逆算子合成策略，用于重塑任务，主要针对结构变换，这可能会限制合成数据的多样性。我们的合成框架在范围上有所不同，它会生成算子流水线以及源–目标表对，并进行可逆噪声注入，从而为自主数据准备提供训练信号。

8 结论

本文介绍了 DeepPRep，这是一种由大语言模型驱动的自主数据准备智能体系统。为解决现有大语言模型驱动方法的局限性，DeepPRep 将数据准备组织为与环境进行迭代式、基于执行的交互，并引入基于树的智能体推理，以支持结构化探索，并根据运行时反馈进行非局部修订。DeepPRep 还采用渐进式智能体训练框架，逐步培养算子级能力和基于树的推理能力，并辅以数据合成，以提供多样且复杂的自主数据准备任务。大量实验有力地表明，DeepPRep 始终优于现有开源基线；在推理成本显著更低的情况下，其准确率可媲美强大的闭源模型（例如 GPT-5）。通过发布源代码、数据集以及一套全面的训练模型（参数规模从 0.5B 到 14B），DeepPRep 为自主数据准备提供了实用、经济高效且可部署的解决方案，并为未来基于执行的智能体系统研究公开提供了基础。

参考文献

[1] 2026。技术报告。xxxx/report.pdf[2] Renat Aksitov、Sobhan Miryoosefi、Zonglin Li、Daliang Li、Sheila Babayan、Kavya Kopparapu、Zachary Fisher、Ruiqi Guo、Sushant Prakash、Pranesh Srini-vasan 等。2023。Rest meets react：用于多步推理大语言模型智能体的自我改进。arXiv 预印本 arXiv:2312.10003（2023）。[3] Daniel W Barowy、Sumit Gulwani、Ted Hart 和 Benjamin Zorn。2015。FlashRelate：通过示例从半结构化电子表格中提取关系数据。ACM SIGPLAN Notices 50，6（2015），218–228。[4] Chengliang Chai、Kaisen Jin、Nan Tang、Ju Fan、Dongjing Miao、Jiayi Wang、Yuyu Luo、Guoliang Li、Ye Yuan 和 Guoren Wang。2025。面向数据高效机器学习的经济高效缺失值填补。ACM数据库系统汇刊 50，3（2025），1–36。[5] Qixu Chen、Yeye He、Raymond Chi-Wing Wong、Weiwei Cui、Song Ge、Haidong Zhang、Dongmei Zhang 和 Surajit Chaudhuri。2025。Auto-Test：学习语义域约束，实现表格的无监督错误检测。ACM 数据管理会刊 3，3（2025），1–27。[6] Tamraparni Dasu 和 Theodore Johnson。2003。探索性数据挖掘与数据清洗。John Wiley & Sons。[7] Haoyu Dong 和 Zhiruo Wang。2024。面向表格数据的大语言模型：进展与未来方向。载于《第47届国际ACM SIGIR信息检索研究与发展会议论文集》，2997–3000。[8] Qingxiu Dong、Lei Li、Damai Dai、Ce Zheng、Jingyuan Ma、Rui Li、Heming Xia、Jingjing Xu、Zhiyong Wu、Baobao Chang 等。2024。上下文学习综述。载于《2024年自然语言处理实证方法会议论文集》，1107–1128。[9] Meihao Fan、Ju Fan、Nan Tang、Lei Cao、Guoliang Li 和 Xiaoyong Du。2024。AutoPrep：采用多智能体框架的自然语言问题感知数据准备。CoRR abs/2412.10422（2024）。https://doi.org/10.48550/ARXIV.2412.10422 arXiv:2412.10422[10] Meihao Fan、Xiaoyue Han、Ju Fan、Chengliang Chai、Nan Tang、Guoliang Li 和 Xiaoyong Du。2024。面向实体解析的经济高效上下文学习：设计空间探索。载于《2024年IEEE第40届国际数据工程会议（ICDE）》。IEEE，3696–3709。[11] Congcong Ge、Yachuan Liu、Yixuan Tang、Yifan Zhu、Yaofeng Tu 和 Yunjun Gao。2025。MontePrep：一种无需目标数据实例的蒙特卡洛驱动自动数据准备方法。arXiv 预印本 arXiv:2509.17553（2025）。[12] Yuhang Ge、Yachuan Liu、Yuren Mao 和 Yunjun Gao。2025。Text-to-Pipeline：连接自然语言与数据准备流水线。arXiv 预印本 arXiv:2505.15874（2025）。[13] Sumit Gulwani、William R Harris 和 Rishabh Singh。2012。利用示例进行电子表格数据操作。《ACM 通讯》55，8（2012），97–105。[14] Daya Guo、Dejian Yang、Haowei Zhang、Junxiao Song、Ruoyu Zhang、Runxin Xu、Qihao Zhu、Shirong Ma、Peiyi Wang、Xiao Bi 等。2025。Deepseek-r1：通过强化学习激发大语言模型的推理能力。arXiv 预印本 arXiv:2501.12948（2025）。[15] Shibo Hao、Yi Gu、Haodi Ma、Joshua Hong、Zhen Wang、Daisy Wang 和 Zhit- ing Hu。2023。用语言模型进行推理，就是利用世界模型进行规划。载于《2023年自然语言处理实证方法会议论文集》，8154–8173。[16] Yeye He、Xu Chu、Kris Ganjam、Yudian Zheng、Vivek Narasayya 和 Surajit Chaudhuri。2018。Transform-data-by-example（TDE）：一种可扩展的数据转换搜索引擎。VLDB 基金会会刊 11，10（2018），1165–1177。[17] Zhipeng Huang 和 Yeye He。2018。Auto-detect：表格中的数据驱动错误检测。载于《2018年国际数据管理会议论文集》，1377–1392。[18] Binyuan Hui、Jian Yang、Zeyu Cui、Jiaxi Yang、Dayiheng Liu、Lei Zhang、Tianyu Liu、Jiajun Zhang、Bowen Yu、Keming Lu 等。2024。Qwen2.5-coder 技术报告。arXiv 预印本 arXiv:2409.12186（2024）。[19] Zhongjun Jin、Michael R Anderson、Michael Cafarella 和 HV Jagadish。2017。Foofah：通过示例转换数据。载于《2017年ACM国际数据管理会议论文集》，683–698。[20] Eugenie Y Lai、Yeye He 和 Surajit Chaudhuri。2025。Auto-Prep：面向自助式商业智能的数据准备步骤整体预测。arXiv 预印本 arXiv:2504.11627（2025）。[21] Hung Le、Yue Wang、Akhilesh Deepak Gotmare、Silvio Savarese 和 Steven Chu Hong Hoi。2022。Coderl：通过预训练模型和深度强化学习掌握代码生成。《神经信息处理系统进展》35（2022），21314–21328。[22] Boyan Li、Jiayi Zhang、Ju Fan、Yanwei Xu、Chong Chen、Nan Tang 和 Yuyu Luo。2025。Alpha-sql：使用蒙特卡洛树搜索实现零样本文本转SQL。arXiv 预印本 arXiv:2502.17248（2025）。[23] Changlun Li、Chenyu Yang、Yuyu Luo、Ju Fan 和 Nan Tang。2025。结合轻量到强大的大语言模型的弱到强提示，实现高准确率、低成本且可解释的数据转换。VLDB 基金会会刊 18，8 (2025), 2371–2384.[24] Dacheng Li、Shiyi Cao、Tyler Griggs、Shu Liu、Xiangxi Mo、Eric Tang、Sumanth Hegde、Kourosh Hakhamaneshi、Shishir G Patil、Matei Zaharia 等。2025。LLM 能轻松从示范中学习推理：结构而非内容才是关键！arXiv 预印本 arXiv:2502.07374（2025）。[25] Haoyang Li、Shang Wu、Xiaokang Zhang、Xinmei Huang、Jing Zhang、Fuxin Jiang、Shuai Wang、Tieying Zhang、Jianjun Chen、Rui Shi 等。2025。Omnisql：大规模合成高质量文本到 SQL 数据。arXiv 预印本 arXiv:2503.02240（2025）。[26] Jinyang Li、Binyuan Hui、Ge Qu、Jiaxi Yang、Binhua Li、Bowen Li、Bailin Wang、Bowen Qin、Ruiying Geng、Nan Huo 等。2024。LLM 是否已能充当数据库接口？用于大规模数据库基础文本到 SQL 的大型基准。神经信息处理系统进展 36（2024）。[27] Peng Li、Yeye He、Cong Yan、Yue Wang 和 Surajit Chaudhuri。2023。Auto-Tables：无需示例，通过合成多步转换实现表格关系化。VLDB 基金会会刊 16，11（2023），3391–3403。https: //doi.org/10 14778 3611479 3611534。 /。Xue Li 和 Till Döhmen。 。利用代码[28] 2024生成与大语言模型实现高效数据整理。载于《第八届端到端机器学习数据管理研讨会论文集》。62–66。[29] Weizheng Lu、Jing Zhang、Ju Fan、Zihao Fu、Yueguo Chen 和 Xiaoyong Du。2025。面向表格处理的大语言模型：综述。《计算机科学前沿》19，2（2025），192350。[30] Theodoros Rekatsinas、Xu Chu、Ihab F Ilyas 和 Christopher Ré。2017。Holoclean：利用概率推断进行整体数据修复。arXiv 预印本 arXiv:1702.00820（2017）。[31] Yucheng Ruan、Xiang Lan、Jingying Ma、Yizhi Dong、Kai He 和 Mengling Feng。2024。表格数据上的语言建模：基础、技术与演进综述。arXiv 预印本 arXiv:2408.10548（2024）。[32] Ranjan Sapkota、Konstantinos I Roumeliotis 和 Manoj Karkee。2025。氛围编程与代码生成：智能体人工智能的基础与实际影响。arXiv 预印本 arXiv:2505.19443（2025）。[33] 字节跳动 Seed、Jiaze Chen、Tiantian Fan、Xin Liu、Lingjun Liu、Zhiqi Lin、Mingxuan Wang、Chengyi Wang、Xiangpeng Wei、Wenyuan Xu 等。2025。Seed1.5-thinking：通过强化学习推动卓越推理模型的发展。arXiv 预印本 arXiv:2504.13914（2025）。[34] Zhihong Shao、Peiyi Wang、Qihao Zhu、Runxin Xu、Junxiao Song、Xiao Bi、Haowei Zhang、Mingchuan Zhang、YK Li、Yang Wu 等。2024。DeepSeekMath：突破开放式语言模型数学推理的极限。arXiv 预印本 arXiv:2402.03300（2024）。[35] Rishabh Singh。2016。BlinkFill：用于句法字符串转换的半监督示例编程。VLDB 基金会会刊 9，10（2016），816–827。[36] Mia Taylor、James Chua、Jan Betley、Johannes Treutlein 和 Owain Evans。2025。奖励黑客学院：对无害任务进行黑客式利用，会使大语言模型泛化出不对齐行为。arXiv 预印本 arXiv:2508.17511（2025）。[37] Xuezhi Wang 和 Denny Zhou。2024。无需提示的思维链推理。神经信息处理系统进展 37（2024），66383–66409。[38] An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv 等。2025。Qwen3 技术报告。arXiv 预印本 arXiv:2505.09388（2025）。[39] Junwen Yang、Yeye He 和 Surajit Chaudhuri。2021。Auto-pipeline：利用强化学习和搜索按目标合成复杂数据流水线。VLDB 基金会会刊 14，11（2021），2563–2575。[40] Shunyu Yao、Dian Yu、Jeffrey Zhao、Izhak Shafran、Tom Griffiths、Yuan Cao 和 Karthik Narasimhan。2023。思维树：利用大语言模型进行审慎的问题求解。神经信息处理系统进展 36（2023），11809–11822。[41] Shunyu Yao、Jeffrey Zhao、Dian Yu、Nan Du、Izhak Shafran、Karthik Narasimhan 和 Yuan Cao。2023。ReAct：在语言模型中协同推理与行动。载于《国际学习表征会议》（ICLR）。[42] 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文本到 SQL 任务的大规模人工标注数据集。载于《2018 年自然语言处理实证方法会议论文集》。3911–3921。[43] Shaolei Zhang、Ju Fan、Meihao Fan、Guoliang Li 和 Xiaoyong Du。2025。Deep-analyze：用于自主数据科学的智能体大语言模型。arXiv 预印本 arXiv:2510.16872（2025）。[44] Yuxin Zhang、Meihao Fan、Ju Fan、Mingyang Yi、Yuyu Luo、Jian Tan 和 Guoliang Li。2025。Reward-SQL：通过逐步推理和过程监督奖励提升文本到 SQL 的表现。arXiv 预印本 arXiv:2505.04671（2025）。[45] Erkang Zhu、Yeye He 和 Surajit Chaudhuri。2017。Auto-join：利用转换连接表。VLDB 基金会会刊 10，10（2017），1034–1045。