Text2Schema：基于自然语言填补数据库表结构设计的空白

Qin Wang, Youhuan Li*
湖南大学 中国湖南
{秦旺，李友欢}@hnu.edu.cn

Yansong Feng
北京大学 中国北京
fengyansong@pku.edu.cn

Si Chen, Ziming Li, Pan Zhang
湖南大学 中国湖南
{sichen,zimingli,hnuzhangpan}@hnu.edu.cn

Zihui Si, Yixuan Chen
湖南大学 中国湖南
{szh-nine,cyx1218}@hnu.edu.cn

Zhichao Shi, Zebin Huang
湖南大学 中国湖南
{史志超，黄志斌1031}@hnu.edu.cn

Guo Chen, Wenqiang Jin
湖南大学 中国湖南
{guochen,wqjin}@hnu.edu.cn

摘要

没有数据库背景的人通常依靠文件系统或 Excel 等工具来管理数据，这往往会导致数据冗余和不一致。关系数据库具有强大的数据管理能力，但需要用户具备较高的专业技能。尽管已经有很多关于 Text2SQL 的研究，旨在将自然语言自动转换为 SQL 查询以进行数据操作，但所有这些研究都假定数据库模式是预先设计好的。实际上，模式设计本身就需要领域专业知识，而直接从文本需求生成模式的研究尚未有人涉足。在本文中，我们系统地定义了一个新问题，称为 Text2Schema，即将自然语言文本需求转换为关系数据库模式。有了有效的 Text2Schema 技术，用户可以轻松地使用自然语言创建数据库表结构，然后利用现有的 Text2SQL 技术进行数据操作，这极大地缩小了非技术人员与高效、多功能的关系数据库系统之间的差距。我们提出 SchemaAgent，这是一个基于大语言模型的用于 Text2Schema 的多智能体框架。我们通过为智能体分配专门的角色并实现有效协作来模拟手动模式设计的工作流程，以完善各自的子任务。我们还引入了专门用于反思和检查的角色，以及一种创新的传输错误检测与恢复机制，以识别和纠正各个阶段的问题。此外，我们构建并开源了一个包含 381 对需求描述和模式的基准。实验结果表明，我们的方法优于比较工作。

PVLDB 参考文献格式：

秦王、李友欢、冯彦松、陈思、李子明、张潘、司子慧、陈逸轩、石志超、黄泽斌和陈果、金文强。《Text2Schema：基于自然语言填补数据库表结构设计的空白》。《PVLDB》，14（1）：XXX-XXX，2020 年。
doi:XX.XX/XXX.XX

PVLDB 资源可用性：

源代码、数据和/或其他成果已发布在 https://github.com/hnuGraph/LLM4DBdesign 。

1 简介

世界各地数以百万计的用户每天依靠电子表格软件处理数据。然而，面对大型数据集或复杂需求时，这类软件容易出现数据冗余和不一致的问题。尽管关系型数据库具备强大的数据管理能力[10, 11, 18]，但由于学习曲线陡峭，大多数人都难以使用。如图 1 所示，实现用户与数据库的交互通常需要两个步骤：工程师（步骤◯1）设计模式以确定表结构；然后（步骤◯2）开发一个简化的用户界面，将常见的数据操作映射为固定的 SQL 语句，从而让普通用户能够与数据库进行交互。然而，这种模式成本高昂，且由于其操作限制过于僵化而存在明显的局限性。

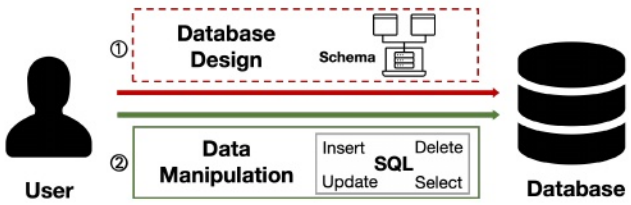

图 1：普通用户与数据库之间的交互。

目前，众多关于 Text2SQL（或 NL2SQL）[3, 47, 48] 的研究旨在将用自然语言表达的数据提取需求转换为可正确执行的 SQL 查询。它们通常从数据库模式中检索一组候选数据列，然后按照 SQL 语法将这些列组织成正确的 SQL 查询语句，对应于步骤 ◯2 图 1 中的。然而，这些工作都隐含了一个前提假设：数据库模式已经设计好并且可以随时用于数据操作。这留下了一个重大问题未得到解决。对于缺乏专业知识的人来说，设计一个稳健的数据库结构（步骤◯1）仍然极其困难，这往往会导致数据库结构混乱以及数据冗余等问题。

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097. doi:XX.XX/XXX.XX

不一致问题。据我们所知，目前还没有研究能够直接从对数据需求的自然语言描述自动生成数据库模式。填补这一研究空白将有助于让非技术用户也能使用强大的数据库系统。

在这项工作中，我们提出了一个名为“Text2Schema”的新问题，该问题专注于直接从用户需求生成数据库模式。我们首次正式定义了数据库模式，包括表结构、主键 - 外键以及域约束。请注意，数据库模式通常表示逻辑模型数据[33]，这与特定于某些数据库产品的数据定义语言（DDL）语句不同。实际上，一旦确定了某个数据库系统，使用辅助工具将数据库模式转换为 DDL 语句是很容易的[8]。我们还展示了一个案例研究，说明了如何将模式转换为一组 SQLite DDL 语句（第 6.8 节）。

与该新问题相关的先前研究可分为两类。第一类是将自然语言转换为可执行的 SQL 查询的 Text2SQL 方法[24]。这些方法不生成数据库模式，无法解决所提出的问题。第二类是自动化数据库设计[7, 25, 32, 39, 42]。它们都专注于使用定制规则或传统深度学习模型实现概念设计的自动化，输出实体和关系。这些方法既不生成数据类型和约束，也不进行逻辑模式中重要的规范化处理。另一种潜在的方法是直接应用大型语言模型（LLMs）[1, 36, 40, 49]来生成模式，因为它们具有强大的推理能力。然而，我们发现这种直接的方法效果不佳。

为解决这一问题，我们提出了一种基于大语言模型的多智能体框架，称为 SchemaAgent，用于自动生成满足第三范式（3NF）[12]的模式，因为在大多数情况下，3NF 模式已足够[26]。具体而言，数据库模式设计主要由三个阶段组成，即（1）用户需求分析；（2）概念设计；（3）逻辑设计（模式）[15, 19]。我们首先为模式生成中的三个子任务分别分配一个智能体（见图 2）：产品经理负责需求分析，概念数据模型设计师和逻辑数据模型设计师。然而，我们发现概念数据建模中的错误率往往较高，这是由于确定适当的实体集、关系和映射基数存在困难。如果在将概念模型映射为表、列或约束之前不纠正这些错误，可能会对模式的质量产生重大影响。因此，我们引入了第四个角色：专门负责及时监督和验证概念模型的审查员。该审查员显著降低了概念模型中的错误率。此外，为了进一步验证逻辑模型是否完整且符合用户需求，我们在 SchemaAgent 中设计了一对质量保证工程师和测试执行者，其中质量保证工程师根据需求生成测试用例，而测试执行者则执行相应的测试以评估模式的质量。

这些角色的顺序工作流程可能会因错误的累积而受到影响，因为我们发现错误往往是在后期才被发现，而不是及时发现。因此，我们设计了一种基于群聊[16, 45]的沟通机制来减少错误

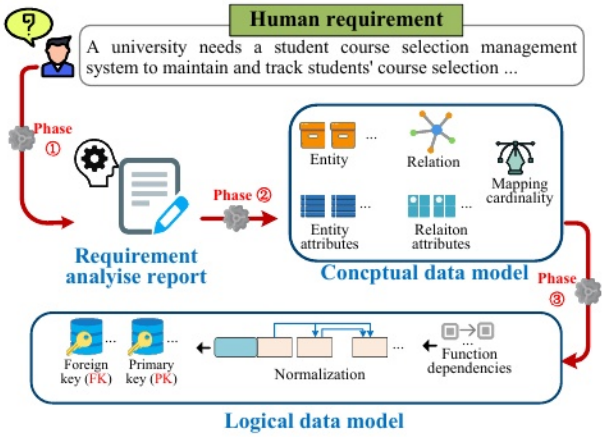

图 2：数据库模式设计流程。第一阶段是需求分析。第二阶段是概念设计阶段，用于构建实体关系（ER）模型。第三阶段是逻辑设计阶段，将 ER 模型映射为逻辑模式。

积累，即工作流程中的各个参与者能够协助识别前一环节所犯的错误。通过精准定位错误发生的阶段并及时反馈，我们的交叉机制引导相关角色改进解决方案。这种动态反馈循环提高了模式生成过程的准确性和效率。

此外，我们开发了首个专门的关系数据库模式基准 RSchema，其中包含 381 对需求及其对应的模式，涵盖了各种现实场景。由于模式生成的复杂性，创建 RSchema 耗时费力，经过了数据库团队多轮精心构建。大量实验表明，我们的方法在 GPT-3.5-turbo、GPT-4o 和 DeepSeek-v3 上均优于链式思维（CoT）和直接提示这两种方法。

我们的贡献总结如下：

- 我们率先提出了 Text2Schema，填补了将自然语言需求转换为数据库模式这一重要空白。我们还给出了数据库模式的首个正式定义。（第 2.2 节）
- 我们率先提出了基于大语言模型的多智能体框架用于文本到模式的转换（第 2 节）。这一新颖的问题也将催生一系列有趣的未来研究方向（第 7 节）。
- 我们在框架中设计了六个角色，并提出了一种可控的传输错误检测与恢复机制，以显著减少累积错误，确保模式质量（第 4 节）。
- 我们创建了首个数据库模式生成基准，其中包含 381 种不同场景下的模式，以及自动评估指标（第 3 节）。
- 实验结果表明，我们的框架显著优于对比方法（第 6 节）。

。

2 初步的

在本节中，我们将讨论数据库模式（第 2.1 节），然后给出其首个正式定义（第 2.2 节）。我们还将回顾相关文献，并从问题和方法两个角度区分我们的工作与以往的研究（第 2.3 节）。

2.1 不同层级的架构

我们主要关注逻辑模式，因为它是设计上的默认数据库模式[33]。在数据库设计中，有三种模式：概念模式（来自概念数据建模）、逻辑模式（来自逻辑数据建模）和物理模式（来自物理设计）。概念模式缺少重要的约束条件，例如外键约束和域约束。此外，它未经过关系规范化，不符合基本的第三范式（3NF）的要求。逻辑模式描绘了表结构和约束（包括数据类型）。它通常需要规范化到 3NF。这是最重要的模式，通常由程序员用于构建应用程序[33]。物理模式隐藏在逻辑模式之下，通常可以轻易更改而不影响应用程序[33]。本文重点在于逻辑模式的实现（以下简称为“模式”）。我们未来的研究将涵盖物理设计，包括针对频繁查询的反规范化以及建立索引以提高查询性能。

2.2 问题定义

逻辑模式是数据库系统中定义数据逻辑组织和约束的结构化组件集合。据我们所知，此前没有研究对数据库模式给出过正式定义。我们在以下定义 1 中对数据库模式进行了正式定义。

定义 1（数据库模式）。数据库（逻辑）模式，记作 S，是一个五元组：S = {R， A， P， F, D}，其中

- R 是 S 内关系（表）标识符的集合。对于每个 $_{R_I}\in$ R， $_{R_I}$is 实质上是一个对应于表的成对元素 ⊠TID，TNAME⊠，其中 $_{TID}$and 和 $_{TNAME}$are 分别对应表的 ID 和表名；
- A 是一组属性（列），其中每个 $_{A_I}\in$A 包含属性 ID（记为 AID）、名称（ANAME）、数据类型（ATYPE），以及 $_{TID}$indicating 属于的表，我们可以用 A（TID） 来表示属于 ID 为 TID 的表的所有属性的集合；
- P 表示主键约束的集合。P 中的每个元素 P_I 恰好包含 ID 为 TID 的唯一表的主键，并且，自然是 A（TID） 的非空子集 $P_I$is；
- F 表示外键约束，每个 $F_I\in$F 是一对属性 ⊠A_{I1}，A I $_2$ ⊠，表示外键属性 AI_1 引用主键属性 $_{AI2}$；
- 最后，D 是域约束的集合，每个 $D_I\in$D 都是一对 ⊠A_I,C $_I$⊠where $A_I$is 一个属性，而 $C_I$indicates 是 {非空、唯一} 中的一个域约束。

这是数据库模式的首个正式定义，我们省略了没有特定或固定形式的用户自定义约束，因为这可能会涉及不确定数量的表或属性。

定义 2（文本转模式）。我们提议研究将业务需求的文本描述转换为数据库模式的问题。我们用 Text2Schema 表示这一新问题，并要求输出模式符合第三范式。

2.3 相关工作

2.3.1 关系数据库设计。自关系数据库问世以来，其设计一直是热门的研究课题。人们开发出了多种方法来解决关系数据库设计中的特定方面的问题。

概念建模被认为是数据库设计过程中最为关键的阶段[38]。在众多方法中，基于文本的方法仍然是最传统和基础的。这些方法可以分为几种不同的类型：基于语言学的（例如 LIDA [5] 和 ER-converter [28]）、基于模式的（例如 APSARA [30]）、基于案例的（例如 CABSYDD [9]）以及基于本体的（例如 OMDDE [34] 和 HBT [39]）。此外，Textodata [7] 提供了一种自动化的解决方案，能够将自然语言文本转换为以 UML 类图表示的目标概念数据库模型。尽管有这些进展，现有的方法往往缺乏深度语义理解，因此在处理复杂场景时表现不佳。

在逻辑设计阶段，概念模型会系统地转换为关系模型。这种转换的基础方法最初由[37]提出。随后，[6]介绍了将增强实体关系（EER）图映射到关系模型的改进策略。

函数依赖（FD）的发现是逻辑数据库模式规范化过程中的一个基本步骤，因为规范化需要识别属性之间的所有函数依赖。FD 发现一直吸引着数据管理领域持续的研究兴趣。[17] 提出了 TANE 算法，该算法通过使用多种剪枝策略来减少搜索空间，从而能够从大型数据库中高效地发现 FD。[23] 提出了一种基于哈希的方法，利用哈希表来检查 FD 的满足情况。[46] 开发了 FastFDs 算法，该算法以深度优先和启发式驱动的方式在搜索树上执行 FD 发现。[29] 提出了 HYFD 混合方法，能够处理具有大量元组和列的数据集。在此基础上，[44] 设计了 DHyFD 动态混合算法，进一步提高了 HYFD 的性能。此外，[41] 提出了 FSC 算法，专门用于大规模数据集中的 FD 发现。上述所有方法都依赖于对现有数据集的分析来提取函数依赖。相比之下，得益于大型语言模型中蕴含的丰富知识，本文旨在在数据库设计过程中自动发现函数依赖关系，从而实现逻辑模式的规范化。

2.3.2 基于 LLM 的多智能体应用。利用 LLM 的单智能体系统通过诸如问题分解[20]、工具利用[22, 52]和内存存储[4]等技术在与环境交互过程中取得了显著进展。在此基础上，多智能体系统通过将 LLM 专门化为特定任务的智能体，并实现自主智能体之间的协作决策，进一步拓展了 LLM 的能力。近期研究突显了多智能体系统在软件开发[14, 16, 45]等领域的成功应用。

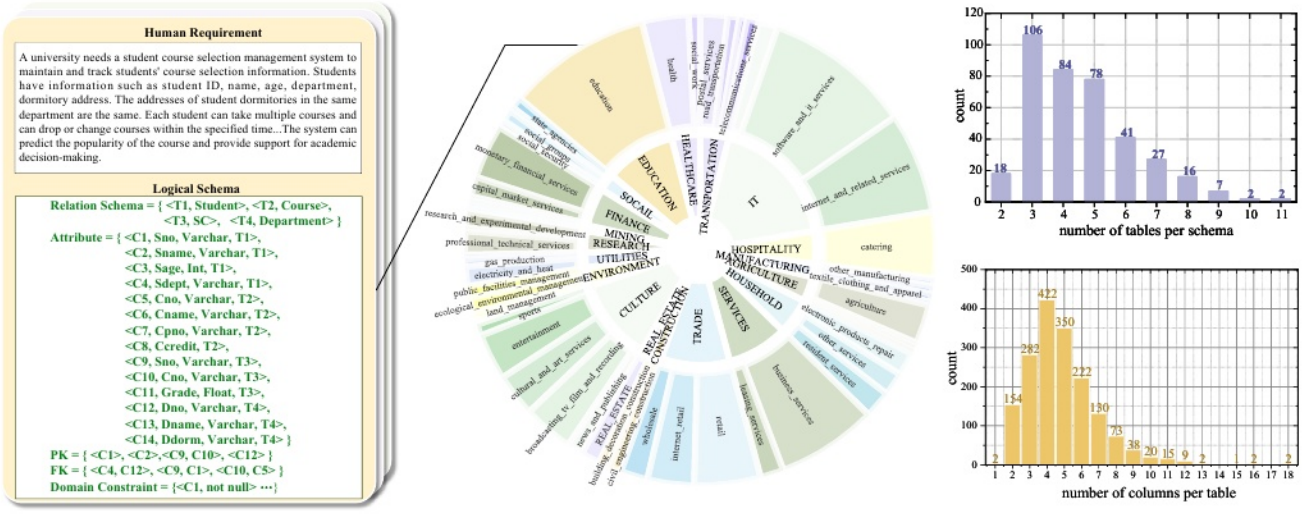

图 3：我们的 RSchema 数据库的结构概览。中间区域展示了领域分布情况。左侧面板展示了一个特定领域派生的示例模式。右上角的面板描绘了每个模式中表的数量分布情况，而右下角的面板则展示了每个表中列的数量分布情况，从而提供了模式复杂性和粒度的洞察。

在数据库领域，多智能体系统主要应用于诸如文本到 SQL 转换[3, 47, 48]、查询优化[43]和数据库诊断[53]等任务。这些应用展示了多智能体系统在提高数据库相关流程的效率、准确性和适应性方面的潜力。

3 语料库构建

我们开发了包含 381 个样本的 RSchema 数据集，涵盖了多个领域。每个样本都由需求文本及其对应的逻辑模式组成。由于隐私方面的顾虑，学术界和工业界都没有公开可用的数据库需求及其对应的模式。据我们所知，这是首个模式生成基准。整个构建过程历时两个月，分为四个阶段，第一阶段是收集原始数据并生成初始样本（第 3.1 节），第二和第三阶段是由数据库小组的 11 名成员进行两轮样本细化（第 3.2 节和第 3.3 节），最后阶段是由一位经验丰富的专家和选定的标注员对所有样本进行审查（第 3.4 节）。图 3 展示了我们数据集的分布情况。总体而言，该数据集包含 19 个主要类别。统计分析表明，平均每个用户需求文本包含 164 个单词，足以表达核心用户需求。

3.1 初始样本生成

我们创建了 500 多个初步样本，这些样本之后会由专家进行完善。这些初步样本是基于从三个不同来源收集的原始数据生成的。第一个来源是包含超过 20,000 个 DDL（数据定义语言）文件的 SchemaPile 数据集 [13]，我们将这些 DDL 内容转换为近 136 个初步模式。我们使用 LLM Qwen2.5-72B-Instruction [49] 为这些 DDL 文件生成相应的初步需求描述。其次，受 LLM 数据生成能力的启发 [35]，我们精心制作了近 93 个涵盖各种行业场景的初步需求描述和模式对。最后，我们还从互联网上抓取的材料（如毕业论文、数据库设计考试、实体关系（ER）图和技术博客）中构建了近 289 个初步样本。

3.2 细化注释

由于原始数据中存在固有的噪声，人工标注对于确保数据质量至关重要。在标注过程中，我们首先对需求文本进行细化，以确保其合理性。接下来，我们遵循严格的数据库设计流程，以获得与需求一致的模式。具体而言，我们识别实体集、关系集和属性等组件，系统地构建概念模型。然后，我们进行依赖关系保持分解并确定键，以构建规范化的逻辑模式。标注人员会遵循设计原则进行详细分析，以保证这些样本的质量。我们采用了四项关键的质量保证措施：

每个需求都应对应一个明确的场景并且

（2）每个模式都应根据相应的需求支持可能的操作；（3）每个模式都必须满足第三范式（3NF），这是约定俗成的[26]；（4）分配给每个标注员的需求平均长度应均衡。每个标注平均需要 25 分钟。

3.3 交叉评审

当对样本进行标注时，会由另一位标注员来验证相应模式的准确性，以确保需求文本的质量。这样可能会出现分歧。

对于同一份样本的不同标注者之间存在的分歧，我们通过增设讨论环节来解决，样本会不断被反复完善，直至标注者达成一致意见。我们还会记录下讨论内容（标注者的意见），作为最终阶段的参考。每次审核平均需要 15 分钟。

3.4 最终审查

最终审核由一位经验丰富的专家和一位选定的标注员共同完成。如果专家和标注员都确认样本质量合格，该样本才会被视为可靠。若双方意见不一致，则由经验丰富的专家做出最终决定。此外，如果专家和标注员都不认可某个样本，该样本将被删除。这一细致的过程耗时一周，最终产生了 381 个样本。

4 方法

4.1 代理结构

我们将标准的数据库模式生成过程划分为若干子任务，每个子任务均由专门的代理负责，如图 4 所示。在我们的 SchemaAgent 框架中，总共有六个角色。这些角色协同工作，遵循系统的数据库设计工作流程来生成全面的数据库模式。SchemaAgent 中的每个基于 LLM 的代理都根据精心设计的配置文件进行操作，该配置文件包括工作描述、目标、约束条件、专业知识和输出格式。所有代理都遵循 [50] 中详细描述的 React 风格的行为。每个角色的配置文件及其相关任务如下所示。

产品经理（PM）代理是直接与用户交互的角色。其主要职责是进行需求分析并生成功能需求分析报告。如清单 1 所示，我们提供了一个静态示例作为演示。该演示涵盖了我们系统的端到端工作流程，每一步的输出都作为相应代理的模板。这种标准化促进了格式的统一，并提高了代理对任务的理解。

```
You are an experienced product manager.
# Goal:
Generate requirement analysis reports: You are
    responsible for analyzing user requirements and
    clarifying any ambiguities by incorporating real-
    world scenarios, ensuring that the requirements are
    clearly defined ...
# Example: {example}
# Input: {input}
```

清单 1：产品经理代理的格式

概念模型设计器（CMD）代理确定概念模型的组件（即实体集、关系集、实体/关系集的映射基数和属性）。清单 2 规定了概念模型设计器的约束条件，要求实体集和关系集之间有明确的区分，因为它们在数据模型中的角色和操作行为本质上是不同的。此外，关系集通常使用组合键而非 ID 来强制执行数据库规范化并避免冗余。

```
You are an expert in building database entity-
    relationship models.
# Goal: Based on the requirements analysis report, define
     the entity sets ... to build a database entity-
    relationship model.
# kownledge:
  - An entity is a "thing" or "object" ...
  - A relationship is a mutual association between
    multiple entities ...
  - The mapping cardinality represents the number of
    other entities ...
# Constraint:
  - Entity set names are mostly nouns, and relationship
    set names are mostly verb or verb-object structures
    ...
  - Most relationship set attributes should not contain
    IDs.
  ...
# Output Format:
  - If you have any uncertainties when identifying
    entities, ... send the issue to the ManagerAgent. If
     you have no questions, the conceptual model design
    is filled in "output".
  - Your final answer is the JSON format converted from
    the entity-relationship model.
# Example: {example}
# Input: {input}
```

清单 2：概念模型设计器代理的格式

概念模型审查器（CMR）代理通过伪代码样式的提示对概念模型提供必要且及时的反馈，以确保全面评估。由于概念模型是整个流程的核心，因此细致的审查至关重要。清单 3 列出了基于伪代码的实体/关系集验证，包括：（1）关系集中不存在冗余 ID，（2）映射基数有效，以及（3）实体引用正确。一旦发现错误，该代理会将检测到的错误及修复建议返回给概念模型设计代理，从而触发改进后的概念模型的重新生成。

```
You are a reviewer of the conceptual model of a database.
# Goal: You will judge whether the conceptual model meets
     its constraints.
# Knowledge: For the conceptual model, you have some
    evaluation criteria described in the form of
    pseudocode. The pseudocode is as follows:
 ```python
   FUNCTION ValidateData(json_data):
    entity_sets = json_data["output"]["Entity Set"]
    relationship_sets = json_data["output"]["
    Relationship Set"]
    # Step 1: Validate Relationship Set
    FOR relationship_name, relationship_details IN
    relationship_sets:
    # 1.1 Check if relationship attributes do not
    contain IDs
    IF Contains_ID_Without_Use(relationship_details["
    Relationship Attribute"]):
      logger.info "Relationship set " +
    relationship_name + " is not standardized:
    Attributes should not contain IDs."
    ...
    logger.info "Validation completed."
   ```
Output Format:
  (1) If the conceptual design does not meet these
    constraints, please send your suggestions to
    ConceptualDesignerAgent.
```

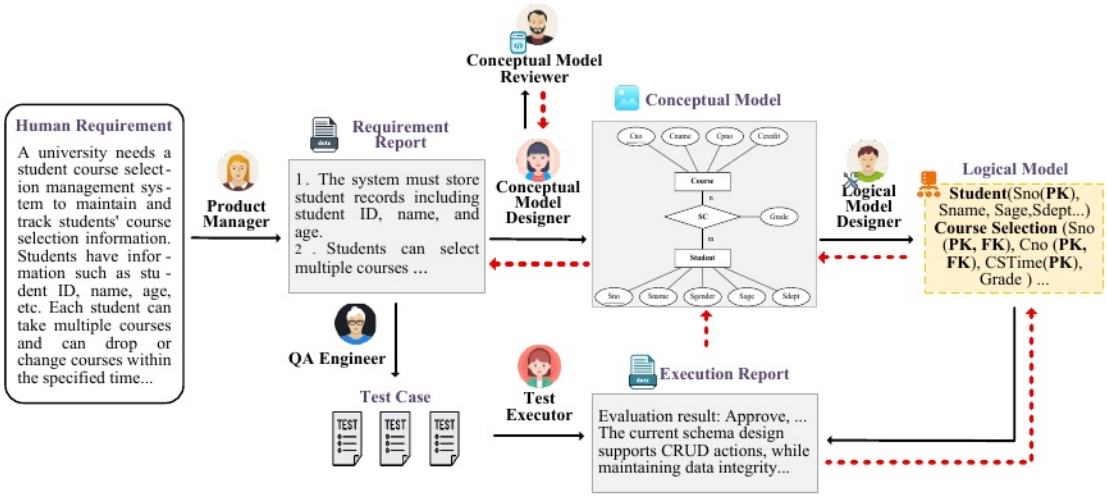

图 4：我们的 SchemaAgent 框架使用基于 LLM 的代理，具有六个不同的角色，包括项目经理、概念模型设计师、概念模型审查员、逻辑模型设计师、质量保证工程师和测试执行者。他们协作处理设计数据库模式所涉及的子任务。红色箭头表示传输错误检测与恢复的过程。

```
  ...
# Example: {example}
# Input: {input}
```

清单 3：概念模型评审代理的格式

逻辑模型设计器（LMD）代理通过识别函数依赖关系和数据类型将概念模型转换为规范化逻辑模式。如清单 4 所示，该代理可能会使用我们封装的工具来识别主键，并基于阿姆斯特朗公理[2]和闭包计算规则对模式进行分解。此过程确保符合 3NF 规范化标准。我们对数据类型采取粗粒度的方法，仅将类别限制为 NUMERIC（数字）、TEXT（文本）、DATETIME（日期时间）、BINARY（二进制）和 BOOL（布尔），而不区分诸如 BIGINT（大整数）或 TINYINT（小整数）等更细的类型。鉴于当前的需求描述主要关注功能需求，非空约束和唯一约束仅应用于主键。随着更详细和明确的需求的出现，将进一步实施其他约束。

```
You are an expert in building the logical model of a
    database.
# Goal: Obtain a database relational schema that conforms
     to the third normal form based on the conceptual
    design of the database.
# Knowledge: {knowledge}
# Constraint:
- Identify functional dependencies and data type in all
    entity sets.
- Use the provided tool to identify the primary keys of
    all entity sets in the conceptual model. If any
    entity set lacks a primary key, the conceptual
    design is deemed invalid. Abort the task and report
    the error to the ConceptualDesignerAgent.
...
# Output format:
- If you find any errors during task execution, you need
    to fill in these errors ...
# Example: {example}
# Input: {input}
```

清单 4：逻辑模型设计器代理的格式

质量保证工程师（QAE）代理根据需求分析报告生成自然语言测试用例。如清单 5 所示，这些用例模拟了诸如插入、删除、更新和查询等实际操作，并包含实际的数据值。

```
You are a quality assurance expert in database design.
# Goal:
According to the requirements analysis, you will generate
     10 sets of test data, each of which includes
    specific values for four operations: insert, delete,
     query, and update.
# Knowledge: {knowledge}
# Constraint:
Your test cases must consider aspects such as entity
    integrity, referential integrity, etc.
# Example: {example}
# Input: {input}
```

清单 5：QA 工程师代理的格式

测试执行器（TE）代理能够理解由质量保证工程师生成的测试用例，并评估所设计的方案是否符合测试标准，最终生成一份全面的测试报告。

```
You are a database expert.
# Gold: You can understand database operations described
    in natural language and judge whether the current
    schemas can meet the operational requirements.
# Constraint:
- If the current schema cannot pass your test, the design
     is unreasonable, and you need to send the error
    report to the role in charge who can solve the
    problem ...
```

```
- If you think it is reasonable after testing, fill with
    "TERMINAL" ...
# Example: {example}
# Input: {input}
```

清单 6：测试执行器代理的格式

经过包括需求分析、概念建模、逻辑模式转换和规范化以及严格测试在内的全面流程，SchemaAgent 最终生成了详细的数据库逻辑模式。

4.2 群组聊天通信机制

我们采用群聊通信机制来实现工作流程[16, 45]，其中各代理（发言者）通过一个由群管理员（本质上是一个大型语言模型）管理的共享消息池相互交流。这种机制能够实现比点对点交互更高效的代理间互动。更重要的是，我们在概念模型设计者（CMD）和概念模型审查者（CMR）之上建立了一个额外的嵌套群组，以促进他们更紧密的沟通。这样一来，与概念模型设计相关的讨论就会被限制在这个嵌套群组内，对其他代理透明。

默认情况下，我们的系统遵循固定的流程，如图 4 中的黑色箭头所示。除了用户输入和需求报告可供所有代理全局访问外，每个代理还维护自己的上下文，其中包括其发送和接收的所有消息。代理生成的输出随后会被存入共享消息池。

4.3 可控错误检测与纠正

我们提出了一种可控的传输错误检测与恢复机制。目前，我们的框架存在两个缺陷。首先，顺序工作流程会随着时间的推移累积错误。其次，固定的发言顺序阻碍了错误反馈的有效传递。实际上，我们发现错误检测可能存在一定的延迟。我们打算让每个代理都能从工作流程中的先前错误中帮助识别可能存在的错误，从而减少累积的错误。

通过这种方式，我们设计了一种机制来控制小组中的发言顺序。具体来说，对于每个角色，我们选择一组其他角色作为可能的下一个发言者。图 4 中的箭头表示这种候选关系，其中黑色箭头构成工作流的顺序部分，而红色虚线箭头用于错误反馈。例如，在 LMD 输出模式时，它可能会检测到概念模型中的错误，因此 CMD 可能是下一个发言者。同样，如果没有错误，模式可以作为输入由 TE 生成执行报告，TE 同样可能是下一个发言者。总体而言，在 LMD 发言（生成模式）之后，有两个可能的下一个发言者，即 CMD 和 TE。图 5 展示了一个 LMD 错误反馈过程的示例。在识别函数依赖关系时，如果 LMD 发现存在没有主键的实体或关系，它会将错误消息发送回 CMD，请求重新生成概念模型。

我们可以通过将这些候选关系整合到代理的个人资料中来实现这一点，这样正在发言的代理就可以根据这些候选关系动态地确定下一个发言者。具体来说，代理在其输出中生成一个标识符，指定其候选列表中的适当代理。该标识符用于确定下一个发言者。如果未选择任何候选者，则预定义的向前发言顺序将作为备用机制来确定下一个发言者。这种机制确保了对话的连贯性和高效推进，同时在多代理交互中允许灵活且基于上下文的决策。

5 设置

5.1 数据集

我们利用本文介绍的 Rschema 数据集，该数据集包含精心构建的涵盖多个领域的数据库设计实例，来有效评估模型在实际数据库设计任务中的性能。

5.2 实施细节

在我们所有的实验中，我们使用 OpenAI 的 API 服务。温度和 top p 均设置为 1.0。当收到包含关键词“TERMINAL”的消息时，群聊会话将终止。鉴于任务本身的复杂性，理论上存在某些实例无法收敛的可能性。为防止此类情况出现无限循环，我们将交互轮次的上限设定为 15 轮。此外，尽管我们实现了多种 JSON 解析机制，但为了防止意外错误，我们设置了三次重试的限制。这一措施确保了所有测试用例都能生成可解析的结果。从统计上看，SchemaAgent 生成单个用例平均需要 35 秒和 0.05 美元。

5.3 评估指标

手动评估每个模式非常耗时，难以对模型性能进行定量评估。因此，我们引入了一种自动评估方法，以 RSchema 中的手动标注模式作为基准。这些模式经过多轮评估，以确保其正确性和优越性。如第 2.2 节所述，一个模式由五个关键部分组成：关系模式（表）、属性、主键、外键和约束。我们计算预测值与基准值在这四个部分上的平均 F1 值和精确匹配准确率（Acc.）。只有当 F1 值等于 1 时，Acc. 才设为 1；否则，Acc. 设为 0。

关系模式（表）由于生成模型可能会生成具有相似含义的不同标记，我们采用三种对齐方法来匹配预测的模式名称与真实值。 (1) 同义词匹配：使用 WordNet [27] 提取预测值的同义词，并验证真实值是否包含在该集合中。 (2) 相似度匹配：利用 all-MiniLM-L6-v2 [31] 测量预测值与真实值之间的语义相似度。阈值 δ_0 设为 0.6。 (3) 字符串匹配：计算预测值与真实值之间的最长公共子串，并检查其长度是否超过阈值 δ_1（设为 0.75）。

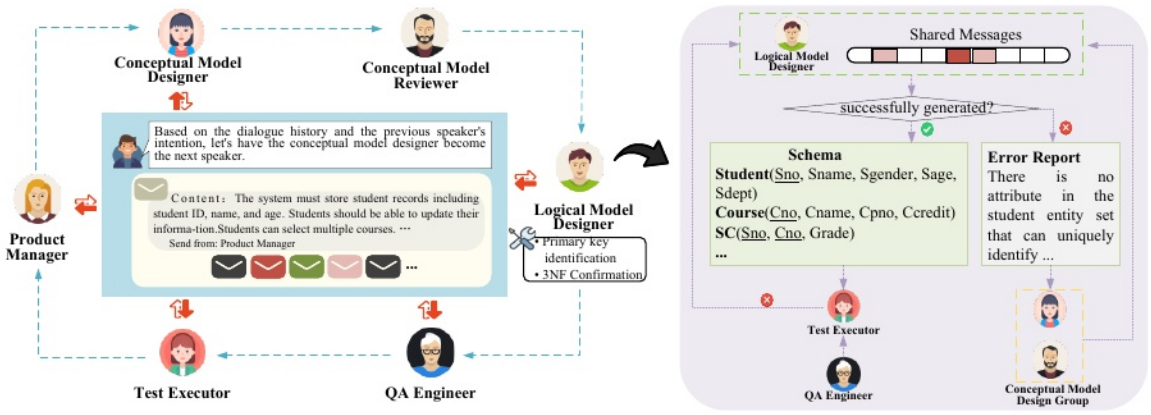

图 5：我们群组聊天通信机制中逻辑模型设计者的错误反馈流程。

在一个模式中有很多表。公式 1 说明了如何为每个模式计算表 F1。

$$\mathrm{F1}_{table} = 2 \times \frac{\mathrm{P} \times \mathrm{R}}{\mathrm{P} + \mathrm{R}} \tag{1}$$

$$\mathrm{P} = \frac{|\mathcal{R}_{gt} \cap \mathcal{R}_{pred}|}{|\mathcal{R}_{pred}|}, \mathrm{R} = \frac{|\mathcal{R}_{gt} \cap \mathcal{R}_{pred}|}{|\mathcal{R}_{gt}|} \tag{2}$$

其中 R_{GT}refers 表示基准模式中的关系模式集合， R_{PRED}refers 表示预测模式中的关系模式集合。

属性 我们还应用上述提到的三种对齐方法来计算映射关系模式中黄金属性集与预测属性集之间的 F1 分数。只有当 F1 等于 1 时，准确率（Acc.）才被设为 1。对于黄金表中没有对应映射到预测表的属性，这些属性的所有指标都被设为 0。由于一个模式包含许多属于不同关系模式的属性，每个样本的属性 F1 计算方法详见公式 3。

$$\mathrm{F1}_{attrs} = \frac{1}{|\mathcal{R}_{gt}|} \sum_{\mathcal{R}_{gt,i} \in \mathcal{R}_{gt}} \mathrm{F1}_{attrs}(\mathcal{R}_{gt,i}) \tag{3}$$

其中 $F1_{ATTRS}(R_{GT,I}$ 是通过将黄金模式 $R_{GT,I}$ 的属性集与预测模式的属性集进行比较来计算得出的。

关键在于，与关系模式和属性不同，我们主张主键和外键要完全匹配。只有当黄金键集与预测键集完全相同时，准确率才为 1 。

数据类型与约束条件 它仅取决于属性，对整体模式结构影响极小。我们仅对成功匹配的属性评估数据类型准确性，未匹配的属性得分为零。由于非空约束和唯一约束仅存在于主键上，因此我们未对约束条件进行评估。

这些指标主要基于语义匹配，不可避免地会包含一些错误。我们对 20 个随机选取的案例进行了人工评估，其中包括超过 100 个表格，评估结果如图 6 所示。如图所示，两种方法之间的差异不具有统计学意义。我们设计的评估指标的平均错误率为 1.8%，在可接受范围内。与人工评估结果的比较证明，我们的评估系统总体上是可靠的，并且能够有效反映模型性能的质量。

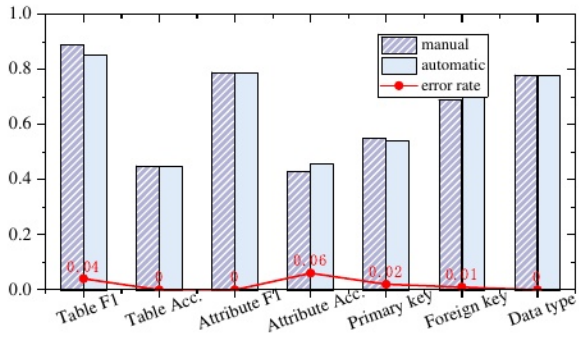

图 6：人工评估和自动评估的结果，以及自动评估的错误率。

5.4 基准线

在 SchemaAgent 框架中，所有代理功能均由相同的大型语言模型（LLM）提供支持，例如 GPT-3.5-turbo、GPT-4o 和 DeepSeekv3。在单样本设置中，一个案例将适用于所有样本。在少样本设置中，我们还额外包含了一个仅涉及两张表的简单案例、一个涉及四张表的中等案例以及一个涉及六张表的复杂案例，以促进上下文学习。在 CoT 设置中，我们手动定义了构建模式的六个关键步骤，分别是：（1）从需求中识别核心实体及其属性；（2）定义实体之间的关系及其基数；（3）将概念模型映射到关系模式和键；（4）定义属性域和完整性约束；（5）对数据库进行规范化以避免冗余；（6）考虑其他必要元素。

6 实验结果与分析

6.1 主要结果

表 1 展示了在 RSchema 基准数据集上的结果。这些指标反映了设计是否满足用户需求。我们在 CoT、一次性学习和少样本学习设置下将 SchemaAgent 与几个主流基线模型进行了比较。从实验结果中我们可以观察到：

表 1：几种方法在 RSchema 数据集上的实验结果。PR. 表示主键。FK. 表示外键。DT. 表示数据类型。

方法	表格		属性		PK	FK.	DT.
	F1	据.....	F1	据.....	据.....	据.....	据.....
GPT-3.5-turbo							
一次性	86.09	51.97	71.65	32.81	53.02	74.01	81.08
一次性+链式思维	86.09	51.71	70.87	33.60	52.49	74.54	81.34
少样本	85.56	49.34	71.13	34.38	54.07	74.80	81.72
模式代理	89.50	61.94	77.69	44.09	64.57	79.27	82.50
GPT-4o							
一次性	85.30	51.18	69.82	35.70	55.12	70.34	82.06
一次性+链式思维	87.14	54.59	71.13	36.48	56.17	71.92	81.94
少样本	88.71	58.79	71.92	35.96	57.74	71.92	82.96
模式代理	90.29	65.09	79.53	49.87	73.23	81.63	83.76
DeepSeek-v3							
一次性	80.58	46.72	67.19	32.55	46.72	63.52	80.08
一次性+链式思维	85.30	49.61	67.98	35.96	46.46	62.20	81.15
少样本	86.88	53.81	72.70	38.06	51.18	62.73	81.60
模式代理	88.98	61.68	78.48	46.72	71.92	80.84	82.35

SchemaAgent 的表现优于其他基于提示的基线模型。我们提出的框架在几乎所有指标上都显著优于相应的基线模型，这表明我们的框架能够生成更高质量的数据库逻辑模式。

CoT 带来了适度的提升。通过 CoT 引入推理步骤，相较于一次性方法，性能略有提高。这表明提示模型“逐步思考”有助于完成识别模式组件这一复杂任务。

在大多数指标上，少样本方法的表现优于链式思维策略。少样本提示方法提供了期望输出及其对应输入的具体示例。少样本学习使模型对任务的理解更加精细，从而能够在更广泛的用例中生成更准确的模式，因此在模式生成任务的各种评估指标上始终优于仅依赖链式思维策略的方法。

骨干模型至关重要。在基于大语言模型的代理框架中，GPT-4o 通常优于 DeepSeek-v3 和 GPT-3.5-turbo，这凸显了代理基础能力的重要性。

6.2 代理角色数量

我们进行了一项消融研究，以更好地理解每个代理的作用。概念模型设计者和逻辑模型设计者是我们框架中的两个基本角色，在任何情况下都不会被移除。我们将其他四个角色作为变量来展示其性能。在移除一个或多个变量角色时，我们始终保留其余代理以保持其原有功能，从而评估性能。如表 2 所示，当所有代理都使用 GPT-4o 时，所有变量角色的加入显著提高了性能，这表明在实现最佳结果方面，专门化角色的重要性。

具体而言，概念模型审查者这一角色最为关键，显著提升了所有指标。这表明概念模型审查者所采用的基于伪代码的验证提示能够有效地识别概念模型中的错误，并提供有价值的反馈以进行修正。质量保证工程师和测试执行者在验证逻辑模型的正确性方面发挥着重要作用。若去掉这两个角色，性能会下降。当只剩下基本的概念模型设计者和逻辑模型设计者时，框架的表现处于最低水平，难以准确提取实体、关系及其相关属性和映射基数。

6.3 代理框架的有效性

我们的框架包含两种错误纠正机制。第一种是嵌套组，由 CMD 和 CMR 组成，专门用于检测和优化概念模型。第二种机制涉及其他代理之间的传输错误检测与恢复。从统计上看，69.9% 的样本收到了错误反馈。其中，60.0% 的错误仅在 CMD 和 CMR 之间被识别和处理。图 7 展示了在不同难度级别下任务中每种类型错误的频率，按表的数量进行分类。它还比较了 SchemaAgent 与次优结果（提示与少量示例）的性能。如图所示，随着表的数量增加，错误反馈的频率稳步上升。特别是，其他错误在所有错误中所占比例也越来越大，这表明随着任务复杂性的增加，潜在错误出现的概率更高。与次优结果相比的改进进一步证明了我们提出的 SchemaAgent 的优越性。

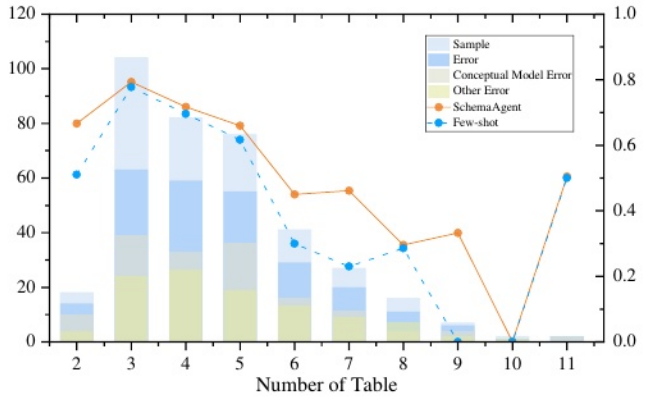

图 7：不同表数量下的错误类型分布及 SchemaAgent 的性能表现。模型性能以“表准确率”分数来表示。

表 2：角色的贡献。“角色”列中从左至右的头像图标分别代表“产品经理”、“概念模型设计师”、“概念模型评审员”、“逻辑模型设计师”、“质量保证工程师”和“测试执行员”。

角色						表格		属性		主键	外键
						F1	据.....	F1	据.....	据.....	据.....
✗	✓	✗	✓	✗	✗	87.14	54.86	70.60	40.68	66.40	67.98
✓	✓	✓	✓	✗	✗	88.71	61.15	74.80	44.09	70.87	80.58
✓	✓	✗	✓	✓	✓	87.66	59.06	74.54	39.37	64.57	78.22
✓	✓	✓	✓	✓	✓	90.29	65.09	79.53	49.87	73.23	81.63
无传输错误检测与恢复											
✓	✓	✓	✓	✓	✓	89.50	62.20	78.22	49.08	71.65	81.10

6.4 可控错误检测与纠正机制的有效性

为了评估所提出的传输错误检测与恢复机制，我们建立了一个基准模型。该模型遵循传统的模式生成流程，除了在概念模型设计期间嵌套组中的反馈外，不包含任何错误反馈。在这个基准模型中，每个代理都尽最大努力完成其指定的任务，然后将输出传递给预定工作流程中的下一个代理。如表 2 所示，在使用可控的错误反馈和恢复机制后，SchemaAgent 的性能得到了提升，所有指标均呈上升趋势。

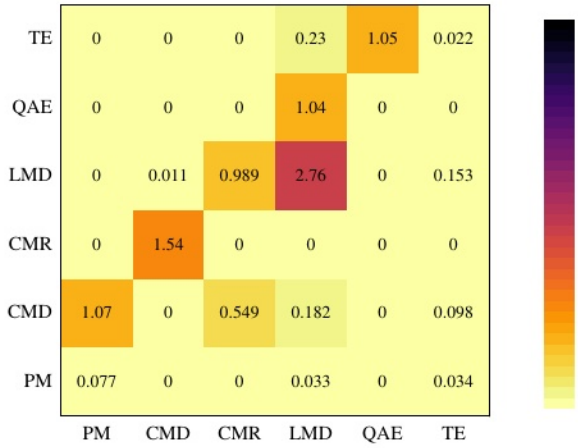

图 8：SchemaAgent 框架中所有代理的通信频率统计。消息从位于横轴的角色传输至位于纵轴的角色。

我们进一步通过分析代理之间的反馈交互频率来研究反馈过程。如图 8 所示，我们可以观察到：（1）CMD 与 CMR 之间的通信最为频繁。这表明在大约一半的情况下，CMR 对当前的概念模型设计提出了挑战。尽管其反馈并非总是完全合理，但这种交互促进了关键的设计细化过程。这也突显了概念模型设计任务的复杂性和关键性。（2）LMD 内部的交互频率相当高。这是由于 LMD 调用了我们封装的工具三次以生成结果：两次用于识别实体集和关系集的主键，一次用于无损依赖分解。这种频率表明 LMD 能够在绝大多数情况下利用工具解决问题。（3）CMD 还会收到 LMD 的错误反馈，而 LMD 的反馈主要来自 TE 和 PM。这些反馈链表明，在我们可控的传输错误检测与恢复机制下，逻辑架构生成的协作效率和准确性得到了提升。由于大型语言模型的不稳定性，一些交互是低频的；后续执行的代理可能仅部分使用先前运行代理的输出（包括其标识符）。这导致了标识符的不可预测性。

6.5 LLM 作为法官

认识到仅依据单一预定义的基准来评估模式设计存在固有的局限性，因为在实际场景中，可能有多种有效的结构都能满足用户需求，因此我们采用另一种基于大型语言模型的方法来进行模式质量的自动评估。具体而言，我们使用 deepseek-r1-0528 作为评判工具。我们从三个关键维度对每个生成的模式进行细致评估，并为每个维度打分，满分为 10 分（从 1 到 10）。这三个维度包括：（1）功能覆盖，评估模式在满足业务需求方面的完整性和可扩展性；（2）数据冗余和规范化，评估其对规范化原则的遵循程度；（3）完整性约束，考察主键及其他关系规则的稳健性。对于每个维度，1 分表示存在根本缺陷，而 10 分则表示完全符合最佳实践（即 100%的功能覆盖、绝对符合第三范式或全面实现完整性）。

图 9 展示了分数分布情况。总分是功能覆盖率、数据冗余与规范化以及完整性约束的加权平均值，其权重分别为 40%、30% 和