Text2Schema: Filling the Gap in Designing Database Table Structures based on Natural Language

Qin Wang, Youhuan Li*
Hunan University
Hunan, China
{qinwang,liyouhuan}@hnu.edu.cn

Yansong Feng
Peking University
Beijing, China
fengyansong@pku.edu.cn

Si Chen, Ziming Li, Pan Zhang
Hunan University
Hunan, China
{sichen,zimingli,hnuzhangpan}@hnu.edu.cn

Zihui Si, Yixuan Chen
Hunan University
Hunan, China
{szh-nine,cyx1218}@hnu.edu.cn

Zhichao Shi, Zebin Huang
Hunan University
Hunan, China
{shizhichao,hzb1031}@hnu.edu.cn

Guo Chen, Wenqiang Jin
Hunan University
Hunan, China
{guochen,wqjin}@hnu.edu.cn

ABSTRACT

People without a database background usually rely on file systems or tools such as Excel for data management, which often lead to redundancy and data inconsistency. Relational databases possess strong data management capabilities, but require a high level of professional expertise from users. Although there are already many works on Text2SQL to automate the translation of natural language into SQL queries for data manipulation, all of them presuppose that the database schema is pre-designed. In practice, schema design itself demands domain expertise, and research on directly generating schemas from textual requirements remains unexplored. In this paper, we systematically define a new problem, called Text2Schema, to convert a natural language text requirement into a relational database schema. With an effective Text2Schema technique, users can effortlessly create database table structures using natural language, and subsequently leverage existing Text2SQL techniques to perform data manipulations, which significantly narrows the gap between non-technical personnel and highly efficient, versatile relational database systems. We propose *SchemaAgent*, an LLM-based multi-agent framework for Text2Schema. We emulate the workflow of manual schema design by assigning specialized roles to agents and enabling effective collaboration to refine their respective subtasks. We also incorporate dedicated roles for reflection and inspection, along with an innovative error detection and correction mechanism to identify and rectify issues across various phases. Moreover, we build and open source a benchmark containing 381 pairs of requirement description and schema. Experimental results demonstrate the superiority of our approach over comparative work.

PVLDB Reference Format:
Qin Wang, Youhuan Li, Yansong Feng, Si Chen, Ziming Li, Pan Zhang, Zihui Si, Yixuan Chen, Zhichao Shi, Zebin Huang, and Guo Chen, Wenqiang Jin. Text2Schema: Filling the Gap in Designing Database Table Structures based on Natural Language. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

*Corresponding author.

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

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/hnuGraph/LLM4DBdesign.

1 INTRODUCTION

Millions of users around the world rely on spreadsheet software for daily data processing. However, when faced with large datasets or complex requirements, such software is prone to data redundancy and inconsistencies. Although relational databases offer powerful data management capabilities [10, 11, 18], they are largely inaccessible to most people due to a steep learning curve. As illustrated in Figure 1, enabling user-database interaction typically involves two steps: engineers (Step ①) design the schema to determine the table structure ; and then (Step ②) develop a simplified UI to map frequent data operations into fixed SQL statements, enabling ordinary users to interact with the database. However, this model incurs significant costs and suffers from evident limitations due to its inflexible operational constraints.

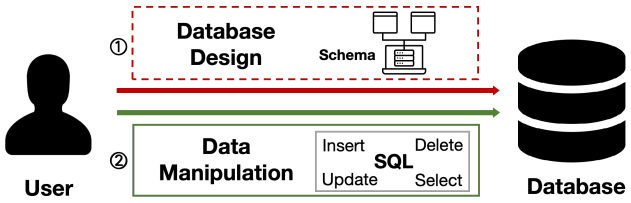

Figure 1: Interactions between ordinary users and databases.

Currently, numerous studies on Text2SQL (or NL2SQL) [3, 47, 48] aim to convert data extraction requirements expressed in natural language into correctly executable SQL queries. They usually retrieve a set of candidate data columns from the database schema and then organize these columns into a correct SQL query statement under the SQL syntax, corresponding to Step ② in Figure 1. However, these works all imply an underlying assumption: the database schema is already designed and readily available for data manipulations. This leaves a significant problem unaddressed. For individuals without expertise, the process of designing a robust database structure (Step ①) is still prohibitively difficult, often leading to poorly structured databases and data redundancy and

inconsistency issues. To the best of our knowledge, there is no research that automatically generates a database schema directly from the natural language description of data requirements. Filling this research gap would be instrumental in empowering non-technical users with powerful database systems.

In this work, we propose a new problem named *Text2Schema*, which focuses on directly generating a database schema from user requirements. We are the first to formally define a database schema, including table structures, primary-foreign keys, and domain constraints. Note that the database schema usually indicates the logical model data [33], which is different from the data definition language (DDL) statements that are specific to certain database products. In fact, it is easy to convert a database schema into DDL statements with auxiliary tools [8] once a certain database system is determined. We also present a case study that demonstrates the conversion of a schema to a set of SQLite DDL statements (Section 6.8).

Previous works related to this new problem could be divided into two categories. The first is Text2SQL that converts natural language into executable SQL queries [24]. These works do not generate database schemas and could not solve the proposed problem. The second is automated database design [7, 25, 32, 39, 42]. All of them focus on the automation of conceptual design with customized rules or traditional deep learning models, outputting entities and relations. These works neither generate data types, constraints, nor conduct normalization that are important for the logical schema. Another potential approach is to directly apply large language models (LLMs) [1, 36, 40, 49] to generate schemas due to their strong reasoning abilities. However, we find that this straightforward approach is poor in effectiveness.

To address this, we propose an LLMs-based multi-agent framework, called *SchemaAgent*, to automatically generate schemas that satisfy the third normal form (3NF) [12], as a 3NF schema is sufficient in most cases [26]. Specifically, database schema design mainly consists of three phases, that is, (1) user requirement analysis; (2) conceptual design; (3) logical design (schema) [15, 19]. We first assign an agent to each of the three subtasks in schema generation (see Figure 2): Product manager for requirement analysis, Conceptual data model designer, and Logical data model designer. However, we find that error rates tend to be high in conceptual data modeling due to the difficulty in determining the appropriate sets of entities, relations, and mapping cardinality. It could have a substantial impact on the quality of the schema if these errors are not corrected before we map the conceptual model into tables, columns, or constraints. Hence, we introduce the 4-th role: a reviewer specifically to supervise and validate the conceptual model in a timely manner. This reviewer significantly reduces errors in the conceptual model. Additionally, to further validate the completeness and correctness of logical model against user requirements, we design a pair of QA engineer and Test executor in SchemaAgent, where the QA engineer produces test cases based on the requirement while the Test executor conducts the corresponding tests to evaluate the schema quality.

The sequential workflow of these roles may suffer from the accumulation of errors, as we find that errors are often discovered later rather than in a timely manner. Therefore, we design a group chat [16, 45] based communication mechanism to reduce error

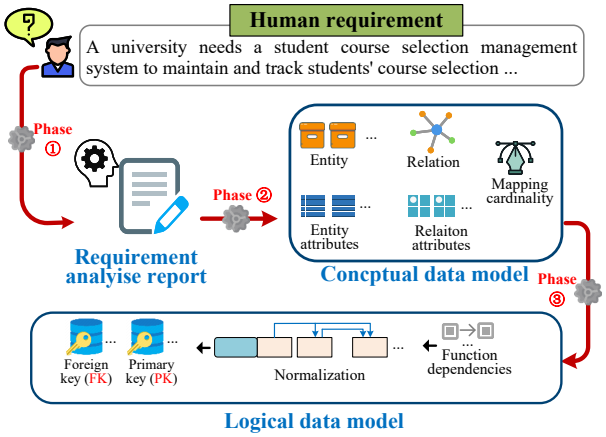

Figure 2: The process for database schema design. The 1st phase is requirements analysis. The 2nd is conceptual design phase for an Entity-Relationship (ER) model. The 3rd is logical design phase mapping ER model into logical schema.

accumulation, where agents in the workflow could assist in identifying the errors made by previous ones. By pinpointing the exact phase where the error occurred and providing timely feedback, our intersection mechanism guides the relevant roles to refine the solution. This dynamic feedback loop improves both the accuracy and efficiency of the schema generation process.

Furthermore, we develop the first specialized relational database schema benchmark *RSchema*, which contains 381 pairs of requirement and the corresponding schema covering various real-world scenarios. Due to the complexity of schema generation, it is time-consuming and laborious to create Rschema, which has undergone multiple rounds of refined construction by database groups. Extensive experiments confirm that our method outperforms both Chain-of-Thought (CoT) and direct prompting approaches across GPT-3.5-turbo, GPT-4o, and DeepSeek-v3.

Our contributions are summarized as follows:

- We are the first to propose Text2Schema and fill a significant gap in converting natural language requirements into database schemas. We also provide the first formal definition of a database schema. (Section 2.2)
- We are the first to propose an LLM-based multi-agent framework for Text2Schema (Section 2). This novel problem will also give rise to a series of interesting future research directions (Section 7).
- We design six roles in our framework and propose a controllable error detection and correction mechanism to significantly reduce accumulated errors, guaranteeing the schema quality (Section 4).
- We create the first database schema generation benchmark, including 381 schemas in diverse scenarios, as well as automatic evaluation metrics (Section 3).
- The experimental results show that our framework significantly outperforms comparative methods (Section 6).

2 PRELIMINARY

In this section, we would discuss database schema (Section 2.1), and then give it a first formal definition (Section 2.2). We would also discuss the literature and distinguish our work from previous work from both the problem and approach perspectives (Section 2.3).

2.1 Schema of Different Level

We primarily focus on the logical schema, as it is the default database schema by design [33]. There are three types of schema in database design: conceptual schema (from conceptual data modeling), logical schema (from logical data modeling), and physical schema (from physical design). The conceptual schema lacks important constraints, such as foreign key constraints and domain constraints. Additionally, it has not undergone relational normalization and fails to meet the requirements of the essential third normal form (3NF). The logical schema depicts the table structures, constraints (including data types). It usually requires normalization to be in 3NF. It is the most important one, which is generally used by programmers to construct applications [33]. The physical schema is hidden beneath the logical one and can generally be easily changed without affecting application programs [33]. This paper focuses on the implementation of the logical schema (hereafter referred to as "schema"). Our future research would incorporate physical design, encompassing denormalization for frequent queries and the establishment of indexes to enhance query performance.

2.2 Problem Definition

A logical schema is a structured collection of components that defines the logical organization and constraints of data within a database system. To the best of our knowledge, no prior studies have provided a formal definition of the database schema. We formally define the database schema in the following Definition 1.

Definition 1 (Database Schema). *A database (logical) schema, denoted as $\mathcal{S}$, is a 5-tuple: $\mathcal{S} = \{\mathcal{R}, \mathcal{A}, \mathcal{P}, \mathcal{F}, \mathcal{D}\}$, where*

- *$\mathcal{R}$ is a set of identifiers of relations (tables) within $\mathcal{S}$. For each $r_i \in \mathcal{R}$, r_i is essentially a pair $\langle tID, tName\rangle$ corresponding to a table, where tID and tName are the corresponding table ID and table name, respectively;*
- *$\mathcal{A}$ is a set of attributes (columns), where each $a_i \in \mathcal{A}$ contains attribute ID (denoted as aID), name (aName), data type (aType), and tID indicating the table to which a_i belongs; we may use $\mathcal{A}(tID)$ to indicate the set all attributes belonging to the table of ID tID;*
- *$\mathcal{P}$ denotes the set of primary key constraints. Each element p_i in $\mathcal{P}$ exactly contains the primary keys of a unique table of ID tID. And, naturally, p_i is a non-empty subset of $\mathcal{A}(tID)$;*
- *$\mathcal{F}$ denotes foreign key constraints, and each $f_i \in \mathcal{F}$ is a pair of attributes $\langle a_{i_1}, a_{i_2}\rangle$ indicating that foreign key attribute a_{i_1} references the primary key attribute a_{i_2};*
- *finally, $\mathcal{D}$ is the set of domain constraints, and each $d_i \in \mathcal{D}$ is a pair $\langle a_i, c_i\rangle$ where a_i is an attribute while c_i indicates one of the domain constraints in $\{$not null, unique$\}$.*

This is the first formal definition of database schema, and we omit user-defined constraints that do not have a specific or fixed form, since it could involve an indefinite number of tables or attributes.

Definition 2 (Text2Schema). *We propose to study the problem of converting text descriptions of business requirements into a database schema. We use Text2Schema to denote this new problem, and we require the output schema to be in 3NF.*

2.3 Related Work

2.3.1 Relational Database Design. Since the advent of relational databases, their design has remained a popular research. A multitude of methodologies has been developed to tackle specific aspects of relational database design.

Conceptual modeling is acknowledged as the most pivotal stage in the database design process [38]. Among the various approaches, text-based methods remain the most traditional and foundational. These methods can be categorized into several distinct types: linguistics-based (e.g., LIDA [5] and ER-converter [28]), pattern-based (e.g., APSARA [30]), case-based (e.g., CABSYDD [9]), and ontology-based (e.g., OMDDE [34] and HBT [39]). Moreover, Textodata [7] offers an automated solution for converting natural language text into a target conceptual database model represented by a UML class diagram. Despite these advancements, existing methods often lack deep semantic understanding, resulting in poor performance when handling complex scenarios.

During the logical design phase, conceptual models are systematically transformed into relational models. The foundational approach to this transformation is first introduced by [37]. Subsequently, [6] introduced improved strategies for mapping Enhanced Entity-Relationship (EER) diagrams to relational models.

The functional dependency (FD) discovery is a fundamental step in the normalization of logical database schemas, as normalization requires identifying all FDs among the attributes. FD discovery has attracted sustained research interest within the data management community. [17] introduced the TANE algorithm, which efficiently discovers FDs from large databases using several pruning strategies to reduce the search space. [23] proposed a hash-based method that leverages hash tables to check FD satisfaction. [46] developed the FastFDs algorithm, which performs FD discovery in a depth-first and heuristic-driven manner over a search tree. [29] proposed a hybrid method, HYFD, which can handle datasets with many tuples and columns. Based on this, [44] designed DHyFD, a dynamic hybrid algorithm that further improves the performance of HYFD. Additionally, [41] proposed FSC algorithms, which are tailored for FD discovery in large-scale datasets. All of the aforementioned methods rely on analyzing existing datasets to extract functional dependencies. In contrast, benefiting from the rich knowledge embedded in LLMs, this paper aims to automatically discover FDs during database design, thereby enabling normalization of the logical schema.

2.3.2 LLM-based multi-agents applications. Single-agent systems that utilize LLMs have achieved notable advances through techniques such as problem decomposition [20], tool utilization [22, 52], and memory storage [4] during environmental interactions. Building on these developments, multi-agent systems have further expanded the capabilities of LLMs by specializing them into task-specific agents and enabling collaborative decision-making among autonomous agents. Recent research highlights the success of multi-agent systems in areas such as software development [14, 16, 45]

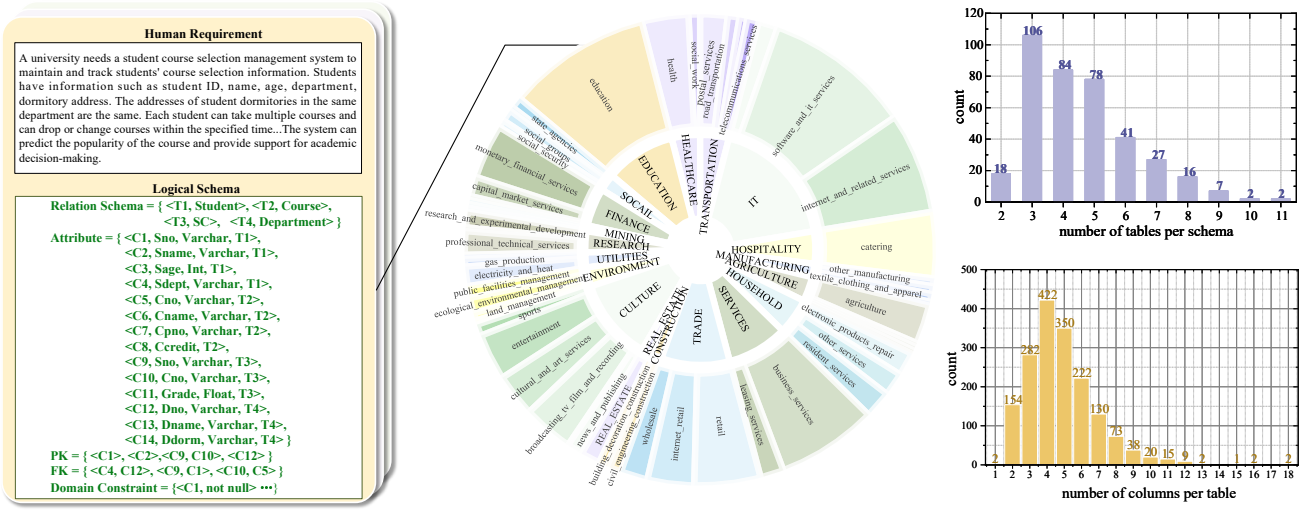

Figure 3: Structural overview of our RSchema database. The central area illustrates the distribution of domains. The left panel presents an example schema derived from a specific domain. The upper right panel depicts the distribution of the number of tables per schema, while the lower right panel shows the distribution of the number of columns per table, providing insights into the schema complexity and granularity.

and biomedical, financial, and psychological domains. In the domain of databases, multi-agent systems have been applied primarily to tasks such as Text-to-SQL [3, 47, 48], query optimization [43], and database diagnostics [53]. These applications demonstrate the potential of multi-agent systems to enhance the efficiency, accuracy, and adaptability of database-related processes.

3 CORPUS CONSTRUCTION

We develop the dataset *RSchema* containing 381 samples covering various domains. Each sample is a pair of requirement text and the corresponding logical schema. Neither academia nor industry has publicly available database requirements and the corresponding schemas due to privacy concerns. To the best of our knowledge, this is the first schema generation benchmark. The entire construction process takes two months, consisting of four phases, where the first is the collection of raw data and the generation of the primary sample (Section 3.1), while the second and third are two rounds of sample refinement by eleven members of a database group (Sections 3.2 and 3.3). The final phase is to review all samples by an experienced expert and a selected annotator (Section 3.4). Figure 3 illustrates the distribution of our dataset. In total, the dataset comprises 19 major categories. Statistical analysis reveals that the average user requirement text contains 164 words, which is sufficient to express the core user requirements.

3.1 Primary Sample Generation

We create more than 500 primary samples that would be refined later by experts. These primary samples are generated on the basis of raw data that are collected from three distinct sources. The first source is the SchemaPile dataset [13] of more than 20,000 DDL (Data Definition Language) files, and we transform these DDL contents into nearly 136 primary schemas. We employ an LLM Qwen2.5-72B-Instruction [49] to generate the corresponding primary requirement descriptions for these DDL files. Secondly, inspired by the data generation capabilities of LLMs [35], we craft nearly 93 primary requirement descriptions and schema pairs that span various industry scenarios. Lastly, we also construct nearly 289 primary samples on materials that are crawled from the Internet, such as graduate theses, database design exams, Entity-Relationship (ER) diagrams, and technical blogs.

3.2 Refinement Annotation

Due to the inherent noise in the raw data, manual annotation is crucial to ensure data quality. During the annotation process, we first refine the requirement text to ensure its reasonableness. Next, we follow the rigorous database design process to obtain a schema that is consistent with the requirements. Specifically, we identify components such as entity sets, relationship sets, and attributes to systematically construct the conceptual model. Then, we conduct dependency-preserving decomposition and identify the keys to build the normalized logical schema. Annotators would conduct detailed analysis obeying design principles, to guarantee the quality of these samples. We employ four key quality assurance measures: (1) each requirement should correspond to an explicit scenario and (2) each schema should support possible operations according to the corresponding requirement; (3) each schema must satisfy the third normal form (3NF), which is a convention [26]; (4) the average length of requirements assigned to each annotator is well balanced. Each annotation takes 25 minutes on average.

3.3 Cross Review

When a sample is annotated, the accuracy of the corresponding schema would be verified by another annotator to ensure the quality of the requirement text. In this way, disagreements may occur

among different annotators for the same sample. We resolve this by setting additional discussions, and the sample would be repeatedly refined until the annotators reach consensus. We also record the discussions (opinions of annotators) as a reference for the final phase. Each review takes 15 minutes on average.

3.4 Final Review

The final review is conducted by an experienced expert and a selected annotator. A sample would be considered reliable if both the expert and the annotator confirm its quality. If there is a disagreement, the experienced expert will make the final decision. In addition, a sample would be deleted if neither the expert nor the annotator accepts it. This meticulous process costs one week and ultimately produces 381 samples.

4 METHOD

4.1 Agent Structure

We divide the standard database schema generation process into several subtasks, each of which is managed by a specialized agent, as indicated in Figure 4. There are in total six roles in our SchemaAgent framework. These roles collaboratively adhere to the systematic database design workflow to generate a comprehensive database schema. Each LLM-based agent in SchemaAgent operates according to a meticulously crafted profile that includes job description, goal, constraints, specialized knowledge and output format. All agents adhere to the React-style behavior as detailed in [50]. The profiles of each role, along with their associated tasks, are presented below.

Product Manager (PM) agent is the role that interacts directly with the user. It is mainly responsible for conducting the requirement analysis and generating the functional requirement analysis report. As illustrated in Listing 1, we provide a static example as a demonstration. This demonstration provides end-to-end coverage of our system workflows, with each step's output acting as a template for the respective agent. This standardization promotes uniform formatting and improves the agent's comprehension of tasks.

```
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}
```

Listing 1: Format of Product Manager agent

Conceptual Model Designer (CMD) agent identifies the components (i.e., the entity set, relationship set, mapping cardinality and attributes of the entity/relation set) of the conceptual model. Listing 2 specifies constraints for the conceptual model designer, mandating a clear separation between entity sets and relationship sets, as their roles and operational behaviors in the data model are fundamentally distinct. Additionally, relationship sets typically employ composite keys rather than IDs to enforce database normalization and avoid redundancy.

```
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}
```

Listing 2: Format of Conceptual Model Designer agent

Conceptual Model Reviewer (CMR) agent provides essential and timely feedback on the conceptual model, using pseudocode-styled prompts to ensure thorough evaluation. As the conceptual model stands as the core of the entire process, a meticulous review is critical. Listing 3 outlines pseudocode-based validation for entity/relationship sets, verifying: (1) absence of redundant IDs in relationship sets, (2) valid mapping cardinalities, and (3) correct entity references. Upon identifying errors, the agent returns detected errors with remediation suggestions to the conceptual model designer agent, triggering the regeneration of an improved conceptual model.

```
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.
```

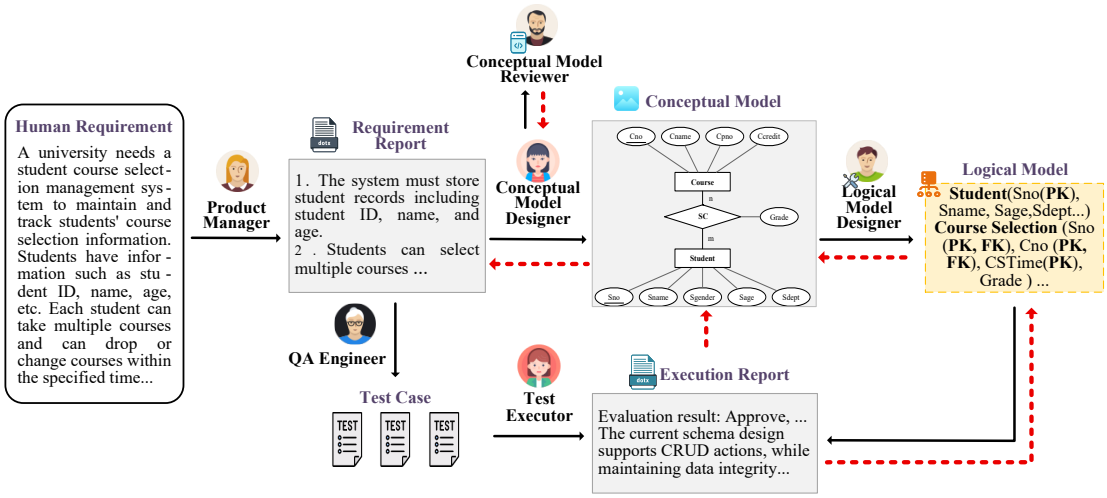

Figure 4: Our *SchemaAgent* framework uses LLM-based agents with six distinct roles, including Project manager, Conceptual model designer, Conceptual model reviewer, Logical model designer, QA engineer, and Test executor. They collaboratively handle the sub-tasks involved in designing the database schema. The red arrow represents the process of error detection and correction.

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

Listing 3: Format of Conceptual Model Reviewer agent

Logical Model Designer (LMD) agent transforms the conceptual model into a normalized logical schema by identifying functional dependencies and data types. As illustrated in Listing 4, this agent may employ our encapsulated tools for primary key identification and schema decomposition based on the Armstrong axioms [2] and the closure computation rules. This process ensures alignment with the 3NF normalization standards. We adopt a coarse-grained approach to data types, limiting categories to NUMERIC, TEXT, DATETIME, BINARY, and BOOL, without distinguishing finer distinctions like BIGINT or TINYINT. Given that the requirement description currently focuses on functional requirements, *not null* constraint and *unique* constraint have been only applied to primary keys. Further constraints will be implemented as more detailed and explicit requirements become available.

```
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}
```

Listing 4: Format of Logical Model Designer agent

QA Engineer (QAE) agent generates natural language test cases based on the requirement analysis report. As illustrated in Listing 5, these cases simulate real-world operations such as insertion, deletion, update, and query, with actual data values.

```
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}
```

Listing 5: Format of QA Engineer agent

Test Executor (TE) agent could understand the test cases generated by the QAE and evaluate if the designed schema meets the testing criteria, culminating in a comprehensive test report.

```
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}
```

Listing 6: Format of Test Executor agent

Following the comprehensive process involving requirement analysis, conceptual modeling, logical schema transformation and normalization, and rigorous testing, SchemaAgent ultimately produces a detailed relational database logical schema.

4.2 Group Chat Communication Mechanism

We implement the workflow using a group chat communication mechanism [16, 45], where agents (speakers) communicate with each other through a shared message pool, managed by a group administrator that is essentially an LLM. This mechanism could achieve an agent interaction that is more efficient than peer-to-peer interactions. More importantly, we establish an additional nested group over the CMD and the CMR to facilitate their closer communication. In this way, discussions related to conceptual model design would be confined within this nested group and transparent to other agents.

By default, our system follows a fixed workflow, as illustrated by the black arrows in Figure 4. In addition to the user input and requirement report, which are globally accessible to all agents, each agent maintains its own context consisting of all messages it has sent and received. The outputs produced by the agents are then deposited in the shared message pool.

4.3 Controllable Error Detection and Correction

We propose a controllable mechanism for error detection and correction. Currently, there are two drawbacks in our framework. Firstly, the sequential workflow tends to accumulate errors over time. Secondly, fixed speaking order hinders the effective transmission of error feedback. In fact, we find that there could be a certain delay in error detection. And we intend to enable each agent to help identify possible errors from previous ones in the workflow, so that we could reduce the accumulated errors.

In this way, we design a mechanism to control the speaker order in the group. Specifically, for each role, we select a set of other roles that could be candidate next speakers. The arrows in Figure 4 indicate such candidate relations, where the black ones constitute the sequential part of the workflow, while the red dotted ones are for error feedback. For example, at the time when the LMD outputs a schema, it may detect errors in the conceptual model, and hence the CMD could be a possible next speaker. Also, if there is no error, the schema could be an input to generate the execution report by the TE, which could similarly be the next speaker. Overall, there are two next candidate speakers, that is, CMD and TE, after the LMD speaks (generating schema). Figure 5 presents an example of the LMD error feedback process. Upon identifying functional dependencies, if the LMD finds that there are entities or relations with no primary keys, it sends the error message back to the CMD, requesting a re-generation of the conceptual model.

We can implement this by integrating these candidate relations into the agent's profile, so that the agent that is speaking can dynamically determine the next speaker from the candidate ones accordingly. Specifically, the agent generates an identifier in its output, designating the appropriate agent from its list of candidates. This identifier allows for the determination of the subsequent speaker. In cases where no candidate is selected, a predefined forward speaking order serves as a fallback mechanism to determine the next speaker. This mechanism ensures that the conversation remains coherent and progresses efficiently, while allowing for flexible and context-aware decision-making in multi-agent interactions.

5 SETTINGS

5.1 Dataset

We utilize the *Rschema* dataset introduced in this paper, which comprises carefully constructed database design instances covering multiple domains, to effectively assess the performance of models in practical database design tasks.

5.2 Implementation Details

In all of our experiments, we use OpenAI's API services. The temperature and top p are set to 1.0. The group chat session is terminated upon the receipt of a message containing the keyword 'TERMINAL'. Given the inherent complexity of the task, there is a theoretical possibility that certain instances may fail to converge. To prevent infinite loops in such cases, we set an upper bound of 15 interaction rounds. Furthermore, while we implement various JSON parsing mechanisms, we configure a retry limit of three attempts to prevent unforeseen errors. This measure ensures that all test cases yield a parsable result. Statistically, SchemaAgent takes an average of 35 seconds and $0.05 to generate a single case.

5.3 Evaluation Metrics

Manually evaluating each schema is prohibitively time-consuming and makes it difficult to quantitatively assess model performance. Consequently, we introduce an automated evaluation approach, using the manually annotated schemas in RSchema as the ground truth. These schemas undergo multiple rounds of evaluation, which ensures their correctness and superiority. As mentioned in Section 2.2, a schema consists of five key components: relation schema(table), attribute, primary key, foreign key, and constraint. We measure the average F1 score and exact matching accuracy (Acc.) between the predicted and ground truth values for these four components. Acc. is set to 1 only if the F1 score equals 1; otherwise, Acc. is set to 0.

Relation schema (Table) Since generative models may produce different tokens with similar meanings, we employ three alignment methods to match predicted schema names with ground truth counterparts. (1) Synonym Matching: WordNet [27] is used to extract synonyms of predicted values and verify the inclusion of the ground-truth value within this set. (2) Similarity Matching: all-MiniLM-L6-v2 [31] is utilized to measure the semantic similarity between predicted and ground-truth values. The threshold δ_0 is set to 0.6. (3) String Matching: We calculate the longest common substring between the predicted and ground-truth values and check if its length exceeds the threshold δ_1 (set to 0.75).

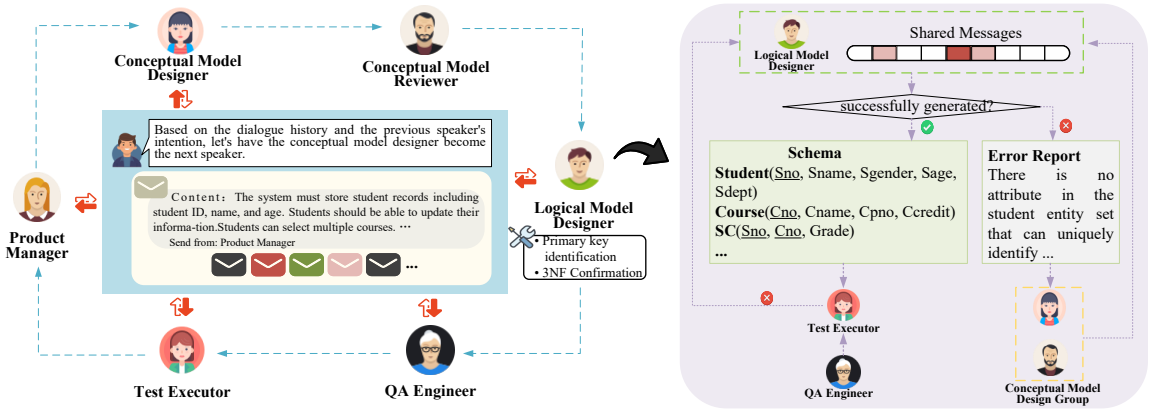

Figure 5: The error feedback process of the logical model designer in our group chat communication mechanism.

There are many tables in a schema. The formula 1 illustrates the computation of table F1 for each schema.

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

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

Where $\mathcal{R}_{gt}$ refers to the set of relation schemas in the ground truth schema, and $\mathcal{R}_{pred}$ refers to the set of relation schemas in the predicted schema.

Attribute We also apply the three alignment methods mentioned above to calculate the F1 score between the golden attribute set and the predicted attribute set in the mapped relation schema. Acc. is set to 1 only if F1 equals 1. For attributes in the golden tables that lack a corresponding mapping to the predicted tables, all metrics for those attributes are set to 0. Since a schema contains many attributes that belong to different relation schemas. The calculation method for attribute F1 of each sample is detailed in formula 3.

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

Where $\mathrm{F1}_{attrs}(\mathcal{R}_{gt,i})$ is computed by comparing the attribute set of the gold schema $\mathcal{R}_{gt,i}$ with that of the predicted one.

Key Unlike relation schema and attributes, we advocate complete matching of primary keys and foreign keys. Acc. is 1 only when the golden key set and the predicted key set are fully identical.

Data type & Constraint It depends solely on attributes with minimal impact on the overall schema structure. We only evaluate data type Acc. for successfully matched attributes, as unmatched attributes yield a score of zero. Due to the fact that *not null* constraints and *unique* constraints are only on the primary key, we have omitted the evaluation of constraints.

These metrics are primarily based on semantic matching and may inevitably include some errors. We perform a manual evaluation of 20 randomly selected cases that includes more than 100 tables, and the results are presented in Figure 6. As depicted in the figure, the discrepancy between the two methods is not statistically significant. The average error rate of our designed evaluation metric is 1. 8%, which is within an acceptable range. The comparison with manual evaluation results proves that our evaluation system is generally reliable and effectively reflects the quality of the model's performance.

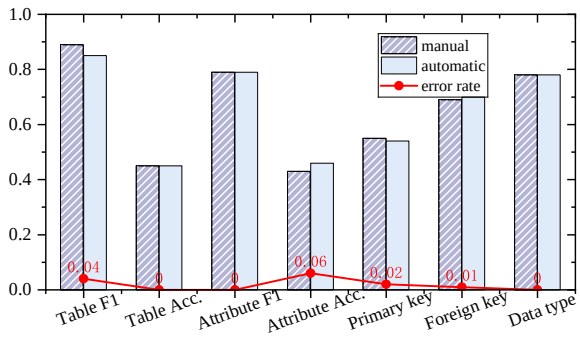

Figure 6: The results of manual evaluation and automated evaluation, as well as the error rate of the automated evaluation.

5.4 Baseline

In the SchemaAgent framework, all agent capabilities are powered by the same LLM, such as GPT-3.5-turbo, GPT-4o, and DeepSeek-v3. In the one-shot setting, a case will run for all samples. In the few-shot setting, we additionally include a simple case with only two tables, a moderate case with four tables, and a complex case with six tables to facilitate context learning. In the CoT setting, we have manually defined six crucial steps for schema construction, namely: (1) Identify core entities and their attributes from requirements; (2) Define relationships between entities and their cardinality; (3) Map conceptual model to relational schemas and keys; (4) Define attribute domains and integrity constraints; (5) Normalize the database to avoid redundancy; (6) Consider other necessary elements.

6 EXPERIMENTAL RESULTS AND ANALYSIS

6.1 Main Result

Table 1 presents the results on the RSchema benchmark dataset. These metrics reflect whether the design meets user requirements. We compare SchemaAgent with several mainstream baseline models in CoT, one-shot, and few-shot settings. We can observe from the experimental results that:

Table 1: Experimental results of several methods on RSchema dataset. PR. refers to primary key. FK. refers to foreign key and DT. refers to data type.

Method	Table		Attribute		PK.	FK.	DT.
	F1	Acc.	F1	Acc.	Acc.	Acc.	Acc.
GPT-3.5-turbo							
One-shot	86.09	51.97	71.65	32.81	53.02	74.01	81.08
One-shot+CoT	86.09	51.71	70.87	33.60	52.49	74.54	81.34
Few-shot	85.56	49.34	71.13	34.38	54.07	74.80	81.72
SchemaAgent	**89.50**	**61.94**	**77.69**	**44.09**	**64.57**	**79.27**	**82.50**
GPT-4o							
One-shot	85.30	51.18	69.82	35.70	55.12	70.34	82.06
One-shot+CoT	87.14	54.59	71.13	36.48	56.17	71.92	81.94
Few-shot	88.71	58.79	71.92	35.96	57.74	71.92	82.96
SchemaAgent	**90.29**	**65.09**	**79.53**	**49.87**	**73.23**	**81.63**	**83.76**
DeepSeek-v3							
One-shot	80.58	46.72	67.19	32.55	46.72	63.52	80.08
One-shot+CoT	85.30	49.61	67.98	35.96	46.46	62.20	81.15
Few-shot	86.88	53.81	72.70	38.06	51.18	62.73	81.60
SchemaAgent	**88.98**	**61.68**	**78.48**	**46.72**	**71.92**	**80.84**	**82.35**

SchemaAgent outperforms other prompt-based baselines. Our proposed framework significantly outperforms the corresponding baseline models mostly across all metrics, demonstrating that our framework could generate higher-quality database logical schemas.

CoT offers modest gains. Introducing a reasoning step via CoT provides a slight performance boost compared to the one-shot method. This suggests that prompting the model to "think step-by-step" aids in the nuanced task of identifying schema components.

Few-shot outperforms the CoT strategy in most metrics. The few-shot prompting method provides concrete examples of desired outputs and their corresponding inputs. Few-shot learning leads to a more refined understanding of the task, enabling the model to generate more accurate schemas across a broader range of use cases, thereby consistently outperforming approaches solely relying on the CoT strategy across various evaluation metrics in the schema generation task.

The backbone model is important. In the LLM-based agent framework, GPT-4o generally outperforms DeepSeek-v3 and GPT-3.5-turbo, highlighting the importance of the foundational capabilities of the agents.

6.2 The Number of Agent Roles

We conduct an ablation study to better understand the contribution of each agent. The conceptual model designer and the logical model designer are two fundamental roles in our framework, which would not be removed in any case. We take the other four roles as variables to demonstrate their performance. Upon removing one or more variable roles, we always keep the remaining agents to retain their original functionality to evaluate performance. As shown in Table 2, when all agents utilize GPT-4o, the inclusion of all variable roles significantly improves performance, indicating the importance of specialized roles in achieving optimal results.

Specifically, the conceptual model reviewer emerges as the most critical role, significantly improving all metrics. This indicates that the pseudocode-based validation prompts employed by the conceptual model reviewer effectively identify errors in the conceptual model and provide valuable feedback for their correction. The QA engineer and test executor play important roles in verifying the correctness of the logical model. Removing these two roles results in a performance decline. When only the fundamental conceptual model designer and logical model designer remain, the framework performs at its lowest level, struggling to accurately extract entities, relations, their associated attributes, and mapping cardinality.

6.3 The Effectiveness of Agent Framework

Our framework incorporates two error correction mechanisms. The first is a nested group, consisting of the CMD and CMR, dedicated to detecting and optimizing the conceptual model. The second mechanism involves error detection and correction among the other agents. Statistically, 69.9% of samples receive error feedback. Of these, 60.0% have errors exclusively identified and handled between the CMD and the CMR. Figure 7 illustrates the frequency of each type of error in varying difficulty levels in the task, categorized by the number of tables. It also compares the performance of SchemaAgent against the suboptimal result (prompt with few-shot). As the figure shows, the frequency of error feedback steadily increases with the number of tables. In particular, *other errors* also account for a growing proportion of all errors, indicating a higher probability of latent errors emerging as task complexity increases. The improvement over the suboptimal result further demonstrates the superiority of our proposed SchemaAgent.

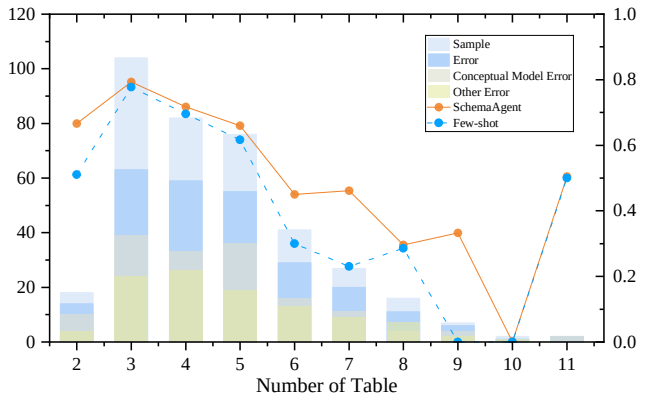

Figure 7: Error type distribution and SchemaAgent performance across varying table counts. The model performance is represented by the "Table Acc." score.

Table 2: The contribution of roles. The avatar icons in the "Roles" column from left to right represent "Product manager", "Conceptual model designer", "Conceptual model reviewer", "Logical model designer", "QA engineer" and "Test executor".

Roles						Table		Attribute		Primary key	Foreign key
						F1	Acc.	F1	Acc.	Acc.	Acc.
✗	✓	✗	✓	✗	✗	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
w/o error detection and correction											
✓	✓	✓	✓	✓	✓	89.50	62.20	78.22	49.08	71.65	81.10

6.4 The Effectiveness of Controllable Error Detection and Correction Mechanism

To evaluate the proposed error detection and correction mechanism, we established a baseline model. This model follows a conventional schema generation process and includes no error feedback, except for the feedback in the nested group during conceptual model design. In this baseline, each agent performs its designated task to the best of its ability and then passes the output to the next agent in a predetermined workflow. As illustrated in Table 2, SchemaAgent exhibits performance improvement after the utilization of the controllable error feedback and correction mechanism, with all metrics demonstrating an upward trend.

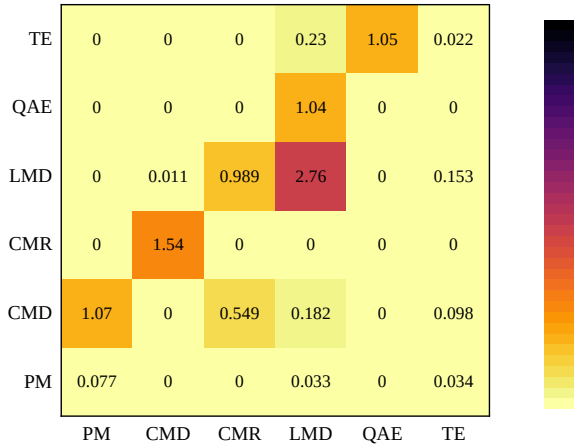

Figure 8: The communication frequency statistics of all agents in the SchemaAgent framework. Messages are transmitted from roles located on the horizontal axis to roles located on the vertical axis.

We further investigate the feedback process by analyzing the frequency of feedback interactions among agents. As depicted in Figure 8, we can observe that: (1) Communication between the CMD and the CMR is the most frequent. This indicates that the CMR challenges the current conceptual model design in roughly half of the cases. Although its feedback may not always be entirely justified, this interaction facilitates a crucial design refinement process. This also highlights the critical and complex nature of the conceptual model design task. (2) The interaction frequency within the LMD itself is pretty high. This is due to the LMD making three additional calls to our encapsulated tools to generate a result: two tool calls for identifying the primary keys of entity and relationship sets, and one for lossless dependency decomposition. This frequency demonstrates that the LMD is able to leverage tools to solve problems in the vast majority of cases. (3) The CMD also receives error feedback from the LMD, whereas the LMD's feedback predominantly originates from the TE and PM. These feedback chains indicate that, under our controllable error detection and correction mechanism, the collaborative efficiency and accuracy of logical architecture generation have been improved. (4) Some interactions are low-frequency due to the instability of LLMs; later-executing agents may only partially use the outputs (including their identifier) of previously running ones. This leads to unpredictable identifiers.

6.5 LLM as The Judge

Recognizing the inherent limitations of evaluating schema designs solely against a single, predefined ground truth, as multiple valid structures could satisfy user requirements in real-world scenarios, we use another LLM-based methodology for automated schema quality assessment. Specifically, *deepseek-r1-0528* is employed as the adjudicator. We provide a granular evaluation of each generated schema across three critical dimensions, assigning a score on a 10-point scale (from 1 to 10) for each. These dimensions encompass: (1) Functional coverage, which assesses the completeness and extensibility of the schema in addressing business requirements; (2) Data redundancy and normalization, evaluating its adherence to normalization principles; and (3) Integrity constraints, examining the robustness of key and other relational rules. For each dimension, a score of 1 indicates fundamental deficiencies, while a score of 10 indicates exemplary adherence to best practices (i.e., 100% functional coverage, absolute compliance with 3NF or comprehensive integrity implementation).

Figure 9 illustrates the score distributions. The overall score is a weighted average of function coverage, data redundancy & normalization, and integrity constraints, with weights of 40%, 30%, and