DBCopilot: Natural Language Querying over Massive Databases via Schema Routing

Tianshu Wang
Institute of Software, CAS
Hangzhou Institute for Advanced Study, UCAS
tianshu2020@iscas.ac.cn

Xiaoyang Chen
University of Chinese Academy of Sciences (UCAS)
Beijing, China
chenxiaoyang19@mails.ucas.ac.cn

Hongyu Lin*
Institute of Software, CAS
Beijing, China
hongyu@iscas.ac.cn

Xianpei Han*
Institute of Software, CAS
Beijing, China
xianpei@iscas.ac.cn

Le Sun
Institute of Software, CAS
Beijing, China
sunle@iscas.ac.cn

Hao Wang
Alibaba Cloud Intelligence Group
Beijing, China
cashenry@126.com

Zhenyu Zeng
Alibaba Cloud Intelligence Group
Hangzhou, China
zhenyu.zzy@alibaba-inc.com

ABSTRACT

The development of Natural Language Interfaces to Databases (NLIDBs) has been greatly advanced by the advent of large language models (LLMs), which provide an intuitive way to translate natural language (NL) questions into Structured Query Language (SQL) queries. While significant progress has been made in LLM-based NL2SQL, existing approaches face several challenges in real-world scenarios of natural language querying over massive databases. In this paper, we present DBCopilot, a framework that addresses these challenges by employing a compact and flexible copilot model for routing over massive databases. Specifically, DBCopilot decouples schema-agnostic NL2SQL into schema routing and SQL generation. This framework utilizes a single lightweight differentiable search index to construct semantic mappings for massive database schemata, and navigates natural language questions to their target databases and tables in a relation-aware joint retrieval manner. The routed schemata and questions are then fed into LLMs for effective SQL generation. Furthermore, DBCopilot introduces a reverse schema-to-question generation paradigm that can automatically learn and adapt the router over massive databases without manual intervention. Experimental results verify that DBCopilot is a scalable and effective solution for schema-agnostic NL2SQL, providing a significant advance in handling natural language querying over massive databases for NLIDBs.

1 INTRODUCTION

Natural Language Interfaces to Databases (NLIDBs) enable users to query and interact with databases by using natural language (NL) rather than complex query languages like Structured Query Language (SQL) [1, 2]. These systems are critical to data democratization by promoting data accessibility and are widely used in practical scenarios such as data analysis and business intelligence. The development of NLIDBs has been greatly advanced with the advent of large language models (LLMs), which provide an intuitive way to translate NL questions into SQL queries via *schema-aware* prompts in a new zero- or few-shot NL2SQL paradigm [22, 25, 36, 38, 45, 55, 58, 66].

However, existing work in this direction typically focuses only on querying a small database with a handful of tables, neglecting real-world scenarios of NLIDBs to query over massive databases (*e.g.,* data lakes, data warehouses, and open data portals) with large-scale schemata [33, 56, 62, 64]. As illustrated in Figure 1, data consumers in this case need to invest significant effort to manually identify and select the appropriate database and relevant tables. Therefore, it becomes imperative to develop a mechanism that abstracts non-experts from the complex data representation and enables them to query over massive databases with schema-agnostic NL questions, *i.e.,* without understanding of large-scale schemata. The core of this process is to construct semantic mappings from diverse NL questions to massive database model elements, thus automatically identifying the target database and filtering for a minimal amount of relevant tables.

Limitations of state-of-the-art. The above scenarios pose significant challenges to current LLM-based NL2SQL, schema linking, and ad-hoc retrieval approaches, including:

C1) Schema Scalability: It is infeasible to feed hundreds or thousands of schemata into LLMs for each question due to token limits and inference costs. Moreover, current LLMs struggle to effectively leverage information present in long contexts [35, 46]. Therefore, numerous studies have employed schema linking as a prerequisite step for LLM-based NL2SQL [38, 55, 70]. However, they follow the typical practice of simultaneously feeding the *entire schema* and NL question into a model [15, 34, 37], which hinders their scalability when dealing with large-scale schemata.

C2) Schema Complexity: Real-world database schemata are characterized by inherent intricacy and heterogeneity in the *relationships* between tables, as shown in Figure 1. These relationships are necessary to identify the correct schema because SQL queries for NL questions often span multiple related tables. However, ad-hoc retrieval methods such as BM25 [75] and DPR [30] for schema linking or retrieval-augmented generation (RAG) [23, 85] merely retrieve elements individually. This approach is suboptimal because the relationships can only be

*Corresponding authors.

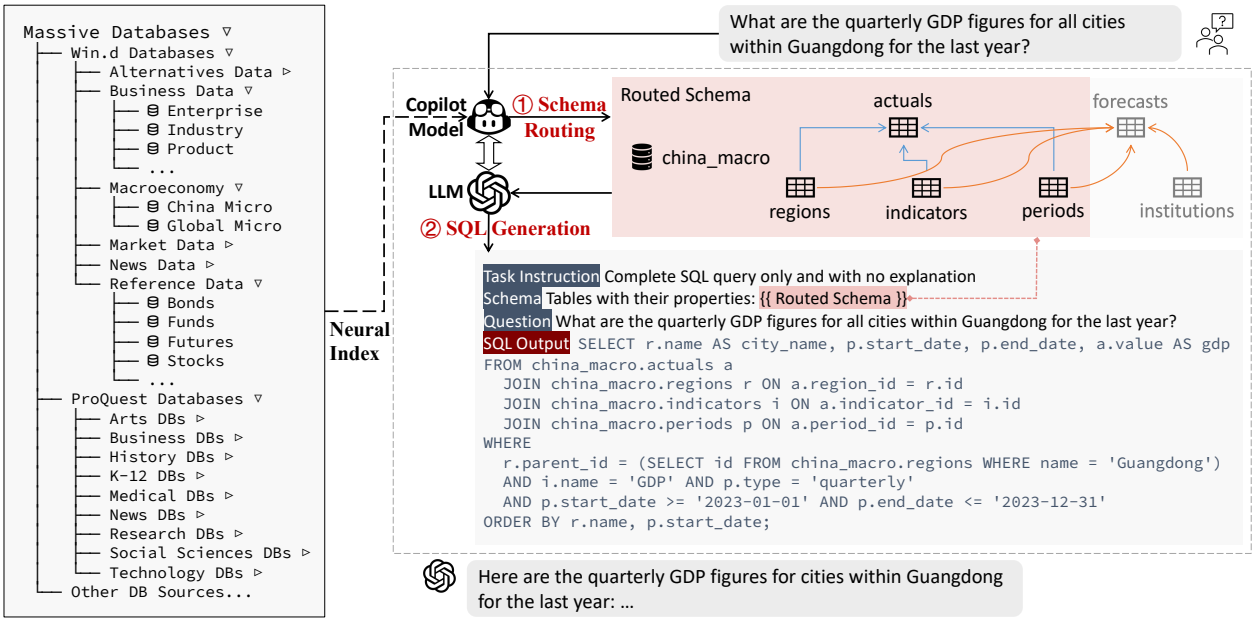

Figure 1: DBCopilot employs LLM-Copilot collaboration to scaling natural language querying over massive databases. Schema routing first navigates to the appropriate database and tables for NL questions via a copilot model. SQL generation then transforms SQL queries via LLMs with schema-aware prompts.

considered through post-processing or re-ranking [8, 33] rather than in an end-to-end manner.

C3) Semantic Mismatch: Unlike general natural language text, database schemata often employ domain-specific or unconventional terminology and naming conventions for tables and columns [20]. This discrepancy entails a *semantic mismatch* between NL questions and schemata. As a result, retrieval-based methods that rely on the co-occurrence of words or sentence representations suffer from the vocabulary mismatch and the diversity of schemata [38, 66]. While some efforts attempt to rewrite NL questions into possible schema elements via LLMs for format alignment [8, 33], they do not adequately address the underlying semantic mismatch.

Approach. In this paper, we present DBCopilot, a framework for effectively scaling natural language querying to massive databases. The main idea behind DBCopilot is to introduce a compact and flexible *copilot* model for routing over massive databases, and to leverage a powerful LLM for generating SQL queries. By decoupling schema-agnostic NL2SQL into schema routing and SQL generation, DBCopilot avoids frequently adapting LLMs to *domain-specific* data while maintaining their *generic* SQL generation capability. As shown in Figure 1, DBCopilot utilizes a lightweight Differentiable Search Index (DSI) [67] to navigate an NL question to its corresponding database and tables. Then the routed schemata and questions are fed into LLMs to generate SQL queries by leveraging their strong capabilities. In this way, DBCopilot can be easily integrated with advanced LLM-based NL2SQL solutions and scale rapidly and cost-effectively to massive databases for NLIDBs.

To effectively implement DBCopilot, we address the above challenges in the following way. For **C1)** *schema scalability*, we leverage generative retrieval (*a.k.a.* DSI) to avoid the need to simultaneously feed whole schemata and the NL question into a neural model. By injecting schema knowledge into the parameters of the schema router, the corresponding SQL query schema can be generated by simply inputting an NL question, thereby extending scalability. For **C2)** *schema complexity*, we propose a relation-aware joint retrieval approach to identify the target databases and tables in an end-to-end manner. We first construct a schema graph to represent the relationships between databases and tables. Based on this graph, we design DFS serialization and constrained decoding algorithms to facilitate relation-aware Seq2Seq retrieval. For **C3)** *semantic mismatch*, we introduce a reverse schema-to-question generation paradigm to automatically synthesize mass training data and efficiently learn the semantic mapping for the schema router. This approach effectively mitigates the labeling cost for massive database schemata and the problem that generative retrieval cannot generalize to unseen schemata during training. Finally, we explore various prompt strategies to select and incorporate multiple candidate schemata for LLM-based SQL generation.

Evaluation. We adapt three standard NL2SQL datasets and two robustness datasets to evaluate DBCopilot. Experimental results verify the effectiveness of DBCopilot, with its copilot model outperforming retrieval-based baselines in schema routing by up to 19.88% in recall, and its schema-agnostic NL2SQL showing an improvement in execution accuracy of 4.43%~11.22%. In summary, we demonstrate that DBCopilot is a scalable and effective solution for schema-agnostic NL2SQL, presenting a significant step forward in handling large-scale schemata of real-world scenarios.

Contributions. Our contributions are summarized as follows.[1]

- We approach the practical and challenging problem of NLIDBs for querying over massive databases, while pointing out the limitations of existing approaches.
- We present DBCopilot, a framework that addresses this problem by decoupling schema-agnostic NL2SQL into schema routing and SQL generation via LLM-copilot collaboration.

[1]We have open-sourced our code and data at: github.com/tshu-w/DBCopilot.

Table 1: Notation list used in this paper.

Symbol	Description
N	Natural language question
Q	SQL query equivalent to question N
$\mathcal{D}$	Set of databases available, each denoted by D
T	Set of tables involved in query Q, with their properties
S	SQL query schema $\langle D, T\rangle$, where T is the subset of tables in database D
$\mathcal{T}$	Union of sets of tables from each database in $\mathcal{D}$
$\mathcal{S}$	Schema space $\mathcal{D} \times \mathcal{P}(\mathcal{T})$[1], where $\mathcal{P}(\mathcal{T})$ is the power set of $\mathcal{T}$
$\mathcal{G}$	Schema graph $\langle \mathcal{V}, \mathcal{E}\rangle$, where $\mathcal{V}, \mathcal{E}$ are graph nodes and edges

[1]The cartesian product of $\mathcal{D}$ and $\mathcal{P}(\mathcal{T})$ is the set of all ordered pairs $\langle D, T\rangle$ that make up the solution (or decoding) space *without any constraints*.

- We propose a relation-aware joint retrieval approach for end-to-end schema routing and a reverse generation paradigm for learning semantic mappings via training data synthesis.
- We conduct thorough experiments to verify the effectiveness of DBCopilot in scaling NL2SQL to massive databases.

2 PRELIMINARIES

2.1 Problem Formulation

As mentioned above, we abstract and formulate the problem of natural language querying over massive databases and large numbers of tables with unspecified SQL query schema (*i.e.,* the target database and tables) as *schema-agnostic NL2SQL*. Formally, as per the notation list detailed in Table 1, the problem of *schema-agnostic NL2SQL* and its two subtasks *schema routing* and *SQL generation* are described in Definition 1, 2, and 3, respectively.

Definition 1 (Schema-Agnostic NL2SQL)**.** Given a natural language question N, schema-agnostic NL2SQL generates an equivalent SQL query Q that can be executed against the appropriate database D drawn from a set of databases $\mathcal{D}$, without being provided with the intended SQL query schema $S = \langle D, T\rangle$.

Definition 2 (Schema Routing)**.** Schema routing identifies the SQL query schema $S = \langle D, T\rangle$ of a natural language question N from the space of all possible schemata $\mathcal{S} = \mathcal{D} \times \mathcal{P}(\mathcal{T})$ within the specified set of databases $\mathcal{D}$ and tables $\mathcal{T}$.

Definition 3 (SQL Generation)**.** SQL generation is the process of generating a SQL query Q, given a natural language question N and the intended SQL query schema $S = \langle D, T\rangle$.

Schema Linking vs. Schema Routing. Schema linking is an optional substep of NL2SQL methods [31]. Given an NL question and a database schema, schema linking aligns natural language terminologies, called "mentions", with their corresponding schema elements (such as tables, columns, and database values). In this way, schema linking servers as a bridge between NL questions and database elements.

Schema routing is an extension of the schema linking concept adapted for querying over massive databases. Specifically, schema routing differs from existing schema linking methods in the following aspects. First, schema linking methods [15, 34, 37] usually take the entire single-database schema and an NL question as input, while schema routing, driven by scalability requirements, should take the NL question as input with massive database schemata indexed. Second, schema routing care about coarse-grained databases and tables, while leaving the fine-grained columns and values for further SQL generation.

2.2 Generative Retrieval

Traditional information retrieval (IR) techniques follow the "index-retrieve" paradigm. Such a pipeline, which cannot be optimized together in an end-to-end manner, limits the final retrieval quality. To address this concern, generative retrieval [67, 72] (*a.k.a.* DSI) is an emerging paradigm that leverages a Seq2Seq pre-trained language model (PLM) to directly generate relevant document IDs (*e.g.,* "doc456") in response to input queries, rather than vector representations for computing similarities. Technically, DSI works by Seq2Seq learning on (query/doc_chunk, doc_id) pairs to model semantic mappings, completely parameterizing traditional "index-retrieve" pipelines within a single neural model. Therefore, unlike conventional machine learning tasks that aim to generalize to unseen data, the training phase of DSI shifts to *memorize* all retrieval targets – including those used in evaluation – similar to the indexing phase of traditional retrieval. Critical ongoing research focuses on defining better document IDs [65, 83]. However, these studies mainly focus on retrieving elements individually without considering their relationships (*e.g.,* joinable relations between tables, triple relations in knowledge graphs, or semantic relations between documents). To the best of our knowledge, this paper is the first to adapt this paradigm to jointly retrieve elements with relationships, opening up new opportunities for applying it in graph-based retrieval [53].

2.3 LLM-based Natural Language to SQL

LLMs, developed by scaling up PLMs in terms of both model and data size, have emerged as a milestone in artificial intelligence [86]. Despite their similar architecture and pre-training tasks, LLMs exhibit unexpected emergent abilities (*e.g.,* in-context reasoning, instruction following) compared to small PLMs [73]. These emergent abilities change the paradigm of model use from "fine-tuning" to "prompt engineering" that does not change the model parameters. This shift has brought new opportunities for all kinds of natural language processing tasks, including NL2SQL. Coupled with well-designed schema-aware prompts and few-shot NL2SQL examples, LLMs can generate SQL queries directly from NL questions, delivering state-of-the-art performance [22, 25, 38]. In addition, pipeline-based approaches decompose NL2SQL into substeps including schema linking, query classification, query decomposition, SQL refinement, further extending the power of LLM-based NL2SQL [43, 55]. Nevertheless, LLM-based NL2SQL still faces significant challenges in real-world scenarios, such as schema scalability, complex or ambiguous queries, SQL efficiency, and so on [20, 33, 40].

3 METHODOLOGY

This section introduces DBCopilot, a schema-agnostic NL2SQL framework for effectively scaling natural language querying to massive databases.

3.1 Overview

As shown in Figure 1, DBCopilot decouples natural language querying over massive databases into schema routing and SQL generation. Specifically, given an NL question N, DBCopilot utilizes a copilot model as a schema router to identify the SQL query schema $S = \langle D, T\rangle$ associated with that question. After that, DBCopilot utilizes an LLM to translate the NL question N with the routed schema S into a corresponding SQL query Q. Example 1 provides an illustration of this process. Consequently, this LLM-Copilot collaboration architecture enables easy integration

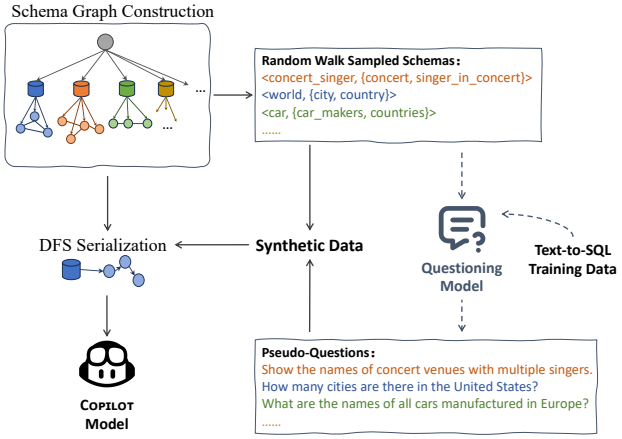

Figure 2: Training process of schema router. We first construct a schema graph to represent the relationships of all databases and their tables. Based on this graph, we sample SQL query schemata and generate pseduo-questions using a reverse-trained schema questioning model that synthesizes training data. Finally, we apply DFS serialization and train the schema router.

with advanced LLM-based NL2SQL solutions and cost-effective scaling to massive databases for NLIDBs.

Example 1. Given a natural language question "*Which singers held concerts in 2022?*", DBCopilot converts it into an equivalent SQL query through the following two phases:
1. Schema Routing identifies the relevant database and tables containing necessary information to answer the question.

⟨concert_singer, {singer, singer_in_concert, concert}⟩

2. SQL Generation prompts LLMs with the identified schema and NL question to generate the SQL query.

```
SELECT s.NAME
FROM concert_singer.singer_in_concert AS sic
JOIN concert_singer.singer AS s
  ON sic.singer_id = s.singer_id
JOIN concert_singer.concert AS c
  ON sic.concert_id = c.concert_id
WHERE c.year = 2022
```

To better scale schema routing to massive databases by modeling relationships between schema elements and learning semantic mappings from natural language to schemata, we propose a relation-aware joint retrieval approach for end-to-end schema routing. Specifically, we first construct a schema graph to represent the underlying relationships between databases and tables (§ 3.2). Based on this graph, we design a schema serialization algorithm to serialize a SQL query schema into a token sequence so that it can be generated by Seq2Seq DSI (§ 3.3). To bridge the semantic gap between natural language and domain-specific schemata, we further propose a reverse generation paradigm for training data synthesis (§ 3.4). These measures allow us to effectively train the schema router with serialized pseudo-instances <N, S> and apply inference with graph-based constrained decoding (§ 3.5). Figure 2 illustrates the training process. Finally, we explore various prompt strategies to select and incorporate multiple candidate schemata for LLM-based SQL generation (§ 3.6).

3.2 Schema Graph Construction

Unlike natural language expressions, a valid SQL query must follow the logical data model [74] of the database. Therefore, it is necessary to inform the schema router about the inherent relationships and constraints between schema elements. In general, the SQL query schema $S = \langle D, T \rangle$ involved in a valid SQL query should satisfy the following conditions. First, the set of tables T should be the subset of tables belonging to database D, since a SQL query should not reference tables that do not exist in the target database. Second, the tables in T should cross-reference at least one other table with a common column (*e.g.*, foreign key), so that the SQL query can connect them together via join or subquery clauses. Example 2 and 3 show two examples. If two tables reference different columns of the junction table, then this junction table is required in a SQL query for these two tables. However, if two tables reference the same column of another table, they can be linked implicitly.

Example 2. Given a natural language question "*What are the names of the singers who performed in a concert in 2014?*", the corresponding SQL query is:

```
SELECT s.name
FROM singer_in_concert AS sc
JOIN singer AS s ON sc.singer_id = s.singer_id
JOIN concert AS c ON sc.concert_id = c.concert_id
WHERE c.year = 2014
```

This query involves the *singer*, *concert*, and *singer_in_concert* tables, where *singer_in_concert* is the join table with two different columns referring to *singer* and *concert* respectively.

Example 3. Given a natural language question "*Which rivers run through the state with the largest city in the US?*", the corresponding SQL query is:

```
SELECT river_name FROM river
WHERE traverse IN (
    SELECT state_name FROM city
    WHERE population = (
        SELECT MAX(population) FROM city
    )
)
```

This query involves only the *city* and *river* tables, both of which refer to the primary key of the *state* table.

To this end, we construct a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ to represent the underlying schemata of all databases $\mathcal{D}$ and their tables $\mathcal{T}$. Concretely, the schema graph consists of three types of nodes $\mathcal{V} = \{v_s\} \cup \mathcal{D} \cup \mathcal{T}$ and many types of edges $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$, where v_s is a special node denoting the *set* of all databases $\mathcal{D}$. Each edge $e \in \mathcal{E}$ in this graph represents a relation between two nodes that can be divided into two categories:

- *Inclusion relations*: Inclusion relations indicate whether a database is part of the database collection and whether a table belongs to a database.
- *Table relations*: Table relations include explicit Primary-Foreign, implicit Foreign-Foreign, and Joinable relations between table nodes. Joinable [60] indicates that two tables share some values in certain attributes, allowing them to be combined together in SQL queries.

Algorithm 1 shows the procedure for constructing this three-tiered hierarchical graph. By capturing the logical structure of database schemata in graph $\mathcal{G}$, any valid single-database SQL

Algorithm 1: Schema Graph Construction.

Input: A dictionary of all database schemata *databases*.
Output: The heterogeneous directed schema graph $\mathcal{G}$.

```
G ← an empty directed graph
for database in databases :
    links ← a dictionary with empty set as default value
    G.addEdge(v_s, database)
    for table in database :
        G.addEdge(database, table)
        // Joinable covers the cases of
           Primary-Foreign and Foreign-Foreign.
        for relatedTable in getJoinableTables(table) :
            G.addEdge(table, relatedTable))
            G.addEdge(relatedTable, table))
return G
```

query schema S is a *trail* [76] on this graph. Each schema trail starts with node v_s, followed by a database node and some of its interconnected table nodes. In this way, the schema graph facilitates relation-based serialization, valid SQL query schema sampling, and graph-based constrained decoding introduced below, serving as the foundation of DBCopilot.

3.3 Schema Serialization

In this paper, we model schema routing as a generative retrieval process from an NL question N to a SQL query schema $\langle D, T \rangle$. Since a Seq2Seq LM takes a sequence of tokens as input and outputs a sequence of tokens, we need to design a schema serialization algorithm that converts the partially ordered SQL query schema into a sequence of tokens.

Basic Serialization. A trivial solution is to randomly order the set of tables as DSI for ad-hoc retrieval [10] and identify the target database based on the votes of the generated tables after inference. However, this serialization loses sight of the inclusion and table relations between schema elements. Due to the position bias and autoregressive nature of Seq2Seq LMs, the order of schema elements is important for training routers to better model their relationships [4, 46]. Therefore, we need a carefully designed serialization strategy that preserves relationships within SQL query schema.

Depth-First Search Serialization. To address the above problem, we follow the constructed schema graph and design a depth-first search (DFS) serialization algorithm. As shown in Algorithm 2, we perform a DFS starting at node v_s until all nodes of a SQL query schema have been visited and record the access order. We then concatenate the sequence of node names into a sequence of tokens. Formally, the *depth-first search serialization* can be defined as:

$$\text{Serialize}_{\pi}(S) \coloneqq \text{Concat}\left([\text{DFS}_{\pi}(S, \mathcal{G})[i]]_{i=1}^{|\text{DFS}_{\pi}(S,\mathcal{G})|}\right)$$

where π is the node iteration order and $\text{DFS}_{\pi}(S, \mathcal{G})$ is the DFS traversal sequence of schema S on schema graph $\mathcal{G}$ when iterating successor nodes under order π and $[i]$ indicates the i-th node of the traversal sequence.

We introduce the iteration order parameter π to determine the DFS sequence because a SQL query schema is not a completely ordered set and there may be multiple DFS sequences for a SQL query schema. We randomly select one DFS sequence at a time as the model target during training, and the schema router assigns different probabilities to different sequences originating from the same schema during inference.

Algorithm 2: Depth-First Search Serialization.

Input: An ordered pair of a SQL query schema $S = \langle D, T \rangle$, the schema graph $\mathcal{G}$, and a node iteration order π.
Output: The serialization of schema S.

```
nodes ← {v_s} ∪ {D} ∪ T
visited ← [ ]                    /* an empty list */
stack ← [v_s]
while stack is non-empty :
    node ← stack.pop()
    visited.append(node)
    if set(visited) is equal to nodes :
        break                    /* cease the loop */
    successors ← [ ]
    // Iterates over successor nodes according
       to node iteration order π
    for successor in G[node] :
        if successor in nodes and not in visited :
            successors.append(successor)
    stack.extend(successors)
return Concat(visited[1 : len(visited)]) /* skip v_s */
```

3.4 Training Data Synthesis

Based on the schema graph and DFS serialization, we can train the schema router on question-schema pairs (N, S) as described in § 3.5. However, it is infeasible to manually label training data for massive database schemata. Furthermore, the generative retrieval paradigm prevents the schema router from generalizing to unseen schemata because they are not injected into model parameters. To address these challenges, we introduce a labor-free training data synthesis method that automatically generates pseudo-instances via a reverse schema-to-question generation paradigm, as illustrated in Figure 2.

Specifically, we sample a large number of valid SQL query schemata from the schema graph and generate a pseudo-question for each sampled schema. These schemata are sampled by performing finite-length random walks starting at node v_s over the schema graph, and the database and tables traversed by a walk form a sampled schema. To generate an NL question for each sampled schema, we train a schema questioning model $\mathcal{M}_{\text{q}}$ in reverse based on the training sets of NL2SQL datasets. The reason for training a specific questioning model rather than using LLMs directly is to save the cost of synthesizing large amounts of training data. We use a SQL parser to extract the metadata (tables and columns) of SQL queries from the training pairs (N, Q). Unlike the schema router, which takes a condensed SQL query schema as output, the schema questioner accepts more detailed schema (*e.g.,* table columns and comments) as input to generate rich and detailed questions. Given an illustration of the model input and output in Figure 3, the training loss of the schema questioning model $\mathcal{M}_q$ with parameters θ_{q} on m extracted schema-question pairs (S, N) is:

$$\mathcal{L}_q = -\sum_{i=1}^{m} \log P\left(N_i \mid S_i; \theta_{\text{q}}\right)$$

In practice, it can also generate high-quality seed data from schemata through LLMs and further boost them in this way.

Pseudo-Question Generation

Input: Ask a question that needs to be answered by combining the contents of all the tables, based on the $\texttt{word}_1$ database schema provided below.

```
- countrylanguage(language, countrycode, official)
  [Optional schema comments, if available.]
- country(code, lifeexpectancy)
```

Output: *What is the official language of the country with the highest life expectancy?*

Figure 3: An illustration of pseudo-question generation.

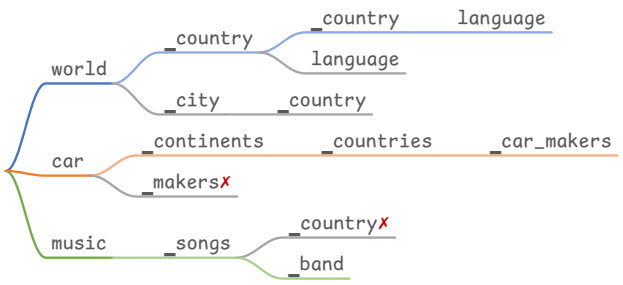

Figure 4: An example of graph-based constrained decoding. The underline (_) denotes the separator between schema elements, and a red cross indicates that the generation of the token is invalid and disallowed. We leverage diverse beam search to ensure a more varied set of candidate sequences, represented in different colors in this graph.

3.5 Schema Router Training & Constrained Inference

Through the above measures, we can efficiently train the schema router to parameterize the semantic mapping of massive database schemata and effectively route NL questions to their target schemata via graph-based constrained decoding. Formally, the Seq2Seq schema router $\mathcal{M}_r$ takes a token sequence of question N as input and generates the serialized schema sequence $\bar{S}$. Suppose the synthetic data is $\{(N_i, S_i) \mid 1 \leq i \leq n\}$, the training loss of schema router $\mathcal{M}_r$ with parameters θ_r is:

$$\mathcal{L}_r = -\sum_{i}^{n} \sum_{\pi} \log P\left(\bar{S}_i \mid N_i; \theta_r\right)$$

After training the schema router, we further design a *graph-based constrained decoding* algorithm to guarantee the validity of generated schemata. As shown in Figure 4, at each step of autoregressive decoding, we maintain a dynamic prefix tree containing the names of accessible nodes from decoded schema elements. Unlike the DFS serialization, the accessible table nodes can only be the neighbors of generated tables, not the neighbors of the database, unless no tables have been generated yet. Therefore, the available tokens (*i.e.,* part of the node name) for each decoding step can be obtained by searching the prefix tree, prefixed with the tokens generated after the last element separator. Besides, multiple candidate schema sequences can be generated to improve the coverage of the target schema. We thus utilize *diverse beam search* [69] to generate more diverse sets of candidate sequences and combine tables from schema sequences that share the same database to form the final candidate schemata.

Basic Prompt

```
### Complete sqlite SQL query only and with no explanation
### Sqlite SQL tables, with their properties:
#
# country(code, name, continent, region, ...)
# countrylanguage(countrycode, language, ...)
#
### Which language is the most popular on the Asian continent?
SELECT
```

Figure 5: An example of a basic NL2SQL prompt.

Chain-of-Thought Prompt

Turn 1:
Based on the provided natural language question, find the database that can best answer this question from the list schemata below. Only output the corresponding database schema identifier in the [id] format, without any additional information.
Question: *Which language is the most popular on the Asian continent?*
Sqlite SQL databases, with their tables and properties:

```
[1] car
continents(contid, continent)
countries(countryid, countryname, continent)
car_makers(id, marker, fullname, country)
[2] word
country(code, name, continent, region, ...)
countrylanguage(countrycode, language, ...)
[3] ... [4] ... [5] ...
```

Turn 2: Filled basic prompt with Turn 1 selected schema.

Figure 6: An example of a chain-of-thought NL2SQL prompt with multiple candidate schemata.

3.6 SQL Generation

By decoupling schema-agnostic NL2SQL into schema routing and SQL generation, DBCopilot can easily integrate with orthogonal, advanced LLM-based NL2SQL solutions, no matter whether in-context prompt engineering methods [7, 22, 26, 55] or specific NL2SQL LLMs [11, 38, 63]. However, since the schema router can generate multiple schemata from different databases for an NL question, we explore three prompt strategies to select and incorporate these candidate schemata for LLM-based NL2SQL:

- **Best Schema Prompting**: This strategy instructs LLMs with the highest probability schema generated by the schema router, as shown in Figure 5. This basic prompt template has been shown to be the most effective prompt for SQL generation with OpenAI LLMs [22].
- **Multiple Schema Prompting**: This strategy instructs LLMs with multiple schemata generated by the schema router for more possible coverage of the target schema. The prompt format is the same as the best schema prompting strategy and we concatenate multiple table schemata in the prompt.
- **Multiple Schema COT Prompting**: This strategy instructs LLMs using multiple candidate schemata through chain-of-thought (COT) reasoning. As shown in Figure 6, the LLM is first asked to select the most relevant candidate schema and then generates SQL queries.

Table 2: Statistics of datasets.

Dataset	Size			Schema		
	Train	Dev.	Test	# DBs	# Tables	# Cols
Spider	7000	–	1034*	166	876	4503
Bird	9427	–	1534*	80	597	4337
Fiben	–	–	279	1	152	374
$\text{Spider}_{\text{syn}}$	7000	–	1034	166	876	4503
$\text{Spider}_{\text{real}}$	–	–	508	166	876	4503

* The original development (dev.) sets are used as test sets.

4 EXPERIMENTS

In this section, we conduct experiments to verify the feasibility and effectiveness of DBCopilot. First, we compare DBCopilot with various retrieval-based methods for schema routing. Then, we evaluate the performance of end-to-end schema-agnostic NL2SQL. Finally, we perform detailed ablation studies.

4.1 Experimental Setup

4.1.1 Research Questions. We conducted experiments to answer the following key research questions (RQs):

RQ1: How does DBCopilot compare with various retrieval baselines in schema routing?

RQ2: Is DBCopilot more robust against semantic mismatch between NL questions and schemata?

RQ3: Does improving schema routing lead to better end-to-end schema-agnostic NL2SQL?

RQ4: What is the best prompt strategy for LLM-based SQL generation when there are multiple candidate schemata?

RQ5: How does each module (*i.e.,* DFS serialization, data synthesis, decoding strategies) contribute to DBCopilot?

4.1.2 Datasets. We adapt existing NL2SQL datasets to evaluate schema routing and schema-agnostic NL2SQL.

- **Spider** [81]: Spider is a widely-used, cross-domain NL2SQL dataset with 10,181 questions and 5,693 unique SQL queries across 138 domains and 200 databases, requiring to generalize across different SQL queries and database schemas.
- **Bird** [40]: Bird is a challenging NL2SQL dataset designed to emphasize the importance of database content, spotlighting the challenges presented by dirty database values, external knowledge requirements, and SQL efficiency.
- **Fiben** [62]: Fiben is a dataset designed to simulate real-world data mart by IBM that closely mirrors actual enterprise data warehouses. Its schema conforms to standard financial ontologies, representing the complexity of schema routing in production environments. Fiben consists of 300 NL questions, corresponding to 237 complex SQL queries. These queries focus on business intelligence scenarios involving nesting and aggregation, making it an ideal testbed for evaluating NL2SQL systems in real-world scenarios.
- **$\text{Spider}_{\text{syn}}$** [21]: $\text{Spider}_{\text{syn}}$ is a dataset designed to evaluate the robustness of NL2SQL models to synonym substitution, built on top of Spider with NL questions modified by replacing schema-related words with real-world paraphrases.
- **$\text{Spider}_{\text{real}}$** [12]: $\text{Spider}_{\text{real}}$ is another robustness dataset created from the dev split of the Spider dataset by manually modifying the original NL questions to remove the explicit mention of column names.

Dataset Adaptation. These datasets are designed to evaluate NL2SQL on a single database, where each question is annotated with its target database. To approach the schema-agnostic NL2SQL, we make several adjustments. First, we remove the single-database constraint and treat all databases in the collection as potential query targets. Next, we extract metadata (tables and columns) from each SQL query using a SQL parser [48] and exclude any queries that cannot be parsed. Finally, we combine the target database and extracted metadata as a SQL query schema, along with the original NL question and SQL query, to form each instance (N, S, Q). This results in three database collections (Spider, Bird, and Fiben), with instances divided into training and test sets as in the original setup. The robustness variants of Spider share the same database collection. Table 2 summarizes the dataset statistics after adaptation.

4.1.3 Baselines. We compare DBCopilot to various sparse and dense retrieval techniques for schema routing (RQ1 & RQ2), including zero-shot, LLM-enhanced, and fine-tuned methods:

- **BM25** [75]: BM25 is a renowned ranking function that serves as a standard in search engines to rank documents by their relevance to a specified query.
- **SXFMR** [59]: SXFMR is a technique for obtaining generic embedding models for dense retrieval by contrastive learning of Transformer(XFMR)-based PLMs on related text pairs.
- **CRUSH** [33]: CRUSH adopts a two-stage approach to retrieve schema elements for NL2SQL. It first instructs an LLM to generate a hallucinated schema, and then combines multiple retrieval results and reranks them based on their relationships to obtain the actual schema.
- **DTR** [27]: DTR is an approach that uses contrastive learning on (question, table) pairs to develop table retrievers for open-domain question answering over tables.

For schema-agnostic NL2SQL (RQ3 & RQ4), since existing NL2SQL methods cannot be directly applied to this problem even when equipped with schema linking (as discussed in § 2.1), we combine a representative LLM-based NL2SQL method [22] with different retrieval methods for comparison. This allows us to isolate and evaluate the effects of schema routing and prompt strategies on end-to-end SQL generation in a controlled setting.

4.1.4 Evaluation Metrics. Different evaluation metrics are used for schema routing and SQL generation respectively.

For *schema routing*, we use Recall@k and mAP (mean average precision) to evaluate the relevance of retrieved databases and tables. Recall@k measures the fraction of relevant instances in the top-k retrieved candidates. mAP calculates the average precision over all queries, which reflects the overall retrieval performance.

For *SQL generation*, traditional surface-level metrics are inappropriate for LLM-based approaches as they may produce SQL queries that differ greatly from the ground truth. Following recent studies [22, 25, 40], we use EX (execution accuracy), which compares the results of generated SQL queries with corresponding ground truths. We also report the cost ($) of LLM invocations.

4.1.5 Implementation Details. We implemented DBCopilot using Python 3.10, Pytorch 2.1, along with some major libraries such as Transformers [77] and Pytorch Lightning [16]. The source code is available at github.com/tshu-w/DBCopilot.

Schema Routing. We adopted T5-base [57] as the backbone of the schema questioner and router. We trained a unified schema questioner on Spider and Bird NL2SQL training sets and three schema routers on each database collection (Spider, Bird, and

Table 3: Schema routing performance on regular test sets.

Method	Spider				Bird				Fiben			
	Database		Table		Database		Table		Database		Table	
	R@1	R@5	R@5	R@15	R@1	R@5	R@5	R@15	R@1	R@5	R@5	R@15
Zero-shot												
BM25	70.12	91.49	86.49	93.87	57.11	90.16	68.32	82.81	98.92	98.92	33.34	38.62
SXFMR	62.57	89.46	80.42	92.39	66.49	88.20	67.56	83.05	100.0	100.0	28.21	46.47
LLM-enhanced												
$\text{CRUSH}_{\text{BM25}}$	72.34	94.49	87.19	95.06	56.58	94.85	68.37	87.82	100.0	100.0	34.87	54.03
$\text{CRUSH}_{\text{SXFMR}}$	63.25	91.97	82.21	93.86	70.14	92.44	70.58	85.07	100.0	100.0	34.06	50.78
Fine-tuned												
BM25	68.28	91.78	86.60	93.87	60.63	90.48	69.87	83.34	98.92	98.92	32.67	38.30
DTR	61.51	92.84	76.27	93.18	76.99	**97.33**	76.24	91.96	100.0	100.0	37.72	48.85
DBCopilot	**85.01**$^{\sharp\ddagger}$	**96.42**‡	**91.63**‡	**97.51**$^{\natural\ddagger}$	**88.92**$^{\sharp\ddagger}$	97.13$^{\sharp}$	**85.83**$^{\sharp\ddagger}$	**94.59**$^{\sharp}$	**100.0**	**100.0**	**41.12**$^{\sharp\dagger}$	**56.89**‡

$^{\natural}$ and $^{\sharp}$ indicate statistically significant improvements over the best LLM-enhanced sparse retrieval baseline (*i.e.,* $\text{CRUSH}_{\text{BM25}}$) at t-test $p < 0.05$ and $p < 0.01$, respectively. † and ‡ indicate statistically significant improvements over the best dense and fine-tuned retrieval baseline (*i.e.,* DTR) at t-test $p < 0.05$ and $p < 0.01$, respectively. The best results are in **bold** and the second best results are underlined. Database recall of BM25 is less than 100% for Fiben due to vocabulary mismatch between some NL questions and the entire database schema, leading to retrieval failures.

Fiben). For each database collection, we sampled 1×10^5 SQL query schemata uniformly from the constructed graph, covering all (100%) databases and tables, and synthesized the corresponding training instances using the schema questioner. We set the batch size to 32, the learning rate to 5×10^{-5}, and the maximum number of training epochs to 20 for all models. Model parameters were optimized using AdamW [47] and a linear learning rate schedule with no warm-up steps. To detect joinable tables besides primary and foreign, we adopted a heuristic way that two tables are joinable if the exact match overlap (Jaccard similarity) of their column values is greater than 0.85. During schema routing, we generated 10 schema sequences for each question using diverse beam search, with beams and beam groups set to 10 and the diversity penalty set to 2.0.

SQL Generation. We used gpt-3.5-turbo-0125[2] to select the optimal schema and generated the SQL query, with the generation temperature set to 0 for reproducibility.

Baselines. To maintain fairness, all baselines compared in this study relied on schema information only. We treated tables as retrieval targets, with the content of each table being the flat normalized names of the table and its columns. For each NL question, we retrieved the top 100 tables and ranked the databases based on the average score of their retrieved tables. We used the standard *Okapi BM25* with two adjustable parameters for BM25 and adopted the best performing *all-mpnet-base-v2* for SXFMR. BM25 and DTR were fine-tuned on synthetic data consistent with DBCopilot to verify our relation-aware joint retrieval.

All experiments were run using Docker image nvidia/cuda:12.1.1-cudnn8-devel-ubuntu22.04 on a server with AMD EPYC 7H12 64-Core Processors, 1TB RAM, and NVIDIA A100 40G GPUs.

4.2 Schema Routing (RQ1 & RQ2)

4.2.1 Experimental Results. We first compare the performance of schema routing on both regular and robustness datasets, with the experimental results shown in Table 3, Table 4, Figure 7, and the following findings.

[2]https://platform.openai.com/docs/models#gpt-3-5-turbo

Table 3 shows the database and table recall on regular test sets. We can see that DBCopilot consistently outperforms all zero-shot, LLM-enhanced, or fine-tuned baselines by a large margin. The database recall@1 of DBCopilot surpasses the second-best approach by up to 12.67% and 11.93% on the Spider and Bird datasets, respectively. This result verifies that DBCopilot can effectively identify the target database of an NL question. Furthermore, DBCopilot also holds an advantage in table recall, with a lead of up to 9.59%, which is crucial for providing complete tables for SQL generation. Even on the single database dataset Fiben, DBCopilot still achieves a 3.4% improvement. These results confirm that DBCopilot can effectively navigate NL questions to their target databases and tables over massive databases.

Finding 1. DBCopilot consistently outperforms all types of baselines by a large margin in schema routing.

When querying over massive databases, data consumers without schema knowledge typically express NL questions that are highly inconsistent with database schemata. To evaluate the robustness of different methods in this situation, Table 4 shows the experimental results on $\text{Spider}_{\text{syn}}$ and $\text{Spider}_{\text{real}}$, which replace explicitly mentioned schema elements in NL questions posed by NL2SQL annotation. We can see that the performance of all methods decreases in this actual scenario setting. Sparse retrieval baselines are significantly affected as they rely heavily on the vocabulary matching, and the LLM-enhanced method also struggles with this challenge, despite being designed specifically for this case. Compared to these baselines, DBCopilot is the least affected and therefore maintains a larger advantage with 14.89% and 19.88% on database recall@1, as well as 2.83% and 6.94% on table recall@5. We believe that by injecting schema knowledge through schema-specific training, DBCopilot can better understand the semantics of database schemata and effectively learn the semantic mapping between NL questions and schemata.

Finding 2. DBCopilot is more robust when NL questions are highly inconsistent with database schemata.

We further analyzed the performance regarding different database sizes (*i.e.,* the number of tables contained) and the number

Table 4: Schema routing performance on robustness tests.

Method	$\text{Spider}_{\text{syn}}$				$\text{Spider}_{\text{real}}$			
	Database		Table		Database		Table	
	R@1	R@5	R@5	R@15	R@1	R@5	R@5	R@15
Zero-shot								
BM25	29.11	58.03	45.30	58.87	52.95	81.1	70.87	80.64
SXFMR	47.78	80.85	65.90	82.96	58.86	88.78	79.48	90.52
LLM-enhanced								
$\text{CRUSH}_{\text{BM25}}$	35.88	65.18	46.80	64.06	58.46	86.42	75.38	89.14
$\text{CRUSH}_{\text{SXFMR}}$	45.45	81.14	67.52	84.18	59.65	91.14	80.99	92.78
Fine-tuned								
BM25	30.66	57.83	46.12	59.65	50.00	81.30	70.77	80.87
DTR	46.52	82.69	65.09	83.19	59.65	89.96	73.23	91.73
DBCopilot	**62.67**$^{\sharp\ddagger}$	**85.11**$^{\sharp}$	**70.35**$^{\sharp\ddagger}$	**86.26**$^{\sharp\ddagger}$	**79.53**$^{\sharp\ddagger}$	**95.47**$^{\sharp\ddagger}$	**87.93**$^{\sharp\ddagger}$	**96.06**$^{\sharp\ddagger}$

We use the same indicators and markups as Table 3.

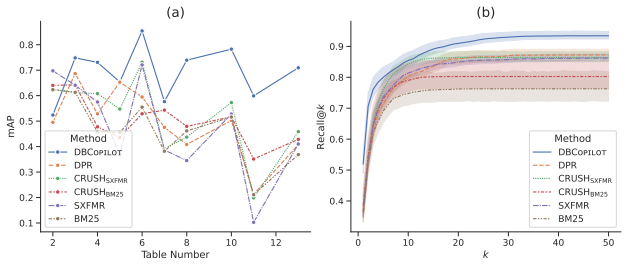

Figure 7: (a) Table mAP *w.r.t.* different numbers of database tables. (b) Table recall@k *w.r.t.* different numbers of retrieved tables.

of tables retrieved. Experimental results are shown in Figure 7 (a) and (b), respectively. First, we categorize the instances based on the number of tables contained in their target database and compare the table mAP across different database sizes. As shown in the Figure 7 (a), the mAP of DBCopilot is relatively stable as the number of database tables increases, while the independent retrieval baselines decrease significantly. We believe this is because by modeling schema routing as a relation-aware joint retrieval process, DBCopilot can better capture the relationships within the database compared to independent retrieval methods, which is critical for routing databases that contain many tables. Furthermore, as shown in Figure 7 (b), the table recall of DBCopilot is consistently higher than that of the baselines as the number of retrieved tables increases, achieving a recall@20 that exceeds the recall@50 of all baselines. This suggests that retrieving more tables and reranking them based on their relationships can hardly compensate for the advantages of our end-to-end relation-aware joint retrieval approach.

Finding 3. Relation-aware joint retrieval are essential for DBCopilot to perform effective schema routing.

4.2.2 Case Study. We provide successful cases to illustrate how DBCopilot improves schema routing through schema-specific training and joint retrieval. As shown in Figure 8, DBCopilot is the only approach that accurately routes the appropriate schema for the given question. Sparse retrieval methods (*i.e.,* BM25 and $\text{CRUSH}_{\text{BM25}}$) rely heavily on lexical matching while ignoring the semantic matching between the question and the schemata. However, untuned dense retrieval methods (*i.e.,* SXFMR and $\text{CRUSH}_{\text{SXFMR}}$) also fail to capture the underlying semantics of

Question: Which language is the most popular on the Asian continent?
SQL Query:

```
SELECT T2.Language
FROM country AS T1 JOIN countrylanguage AS T2
  ON T1.Code = T2.CountryCode
WHERE T1.Continent = "Asia"
GROUP BY T2.Language
ORDER BY COUNT(*) DESC
LIMIT 1
```

SQL Schema: ⟨word, {country, countrylanguage}⟩

BM25	⟨car, {continents, countries}⟩
SXFMR	⟨roller_coaster, {country}⟩
$\text{CRUSH}_{\text{BM25}}$	⟨car, {continents, countries}⟩
$\text{CRUSH}_{\text{SXFMR}}$	⟨match_season, {country}⟩
DTR	⟨world, {countrylanguage}⟩
DBCopilot	⟨world, {countrylanguage, country}⟩

Figure 8: Successful case of DBCopilot in schema routing.

Question: How many flights fly from Aberdeen to Ashley?
SQL Query:

```
SELECT COUNT(*) FROM FLIGHTS AS T1
JOIN AIRPORTS AS T2
  ON T1.DestAirport = T2.AirportCode
JOIN AIRPORTS AS T3
  ON T1.SourceAirport = T3.AirportCode
WHERE T2.City = "Ashley" AND T3.City = "Aberdeen"
```

SQL Schema: ⟨flight, {airports, flights}⟩

DTR	⟨flight, {flights, airports, airlines}⟩
DBCopilot	⟨flight, {flights, airlines}⟩

Figure 9: Failure case where DTR retrieves the correct tables but DBCopilot fails. This is because the schema questioning model incorrectly generates similar but wrong questions for the ⟨$\textbf{flight}_2$, {flights, airlines}⟩ schema.

tables and retrieve the wrong databases. Fine-tuning based approaches identify the correct database where DTR misses the `country` table linked to the `countrylanguage` table, but our relation-aware DBCopilot does not.

While DBCopilot consistently outperforms the baselines, there are a few cases where DTR retrieves the correct schema but DBCopilot does not. For example, Figure 9 shows that DBCopilot confuses the `airlines` and `airports` tables of database `flight`, but DTR covers the correct tables by retrieving more loosely. Upon further analysis, we find that the quality of synthetic data impacts the robustness of schema routing. This is attributed to two main factors: 1) hallucination [52], where the generated NL questions deviate from sampled schemata; 2) generation bias, where the simple pipeline lacks diversity and quality control mechanisms. Developing more sophisticated data synthesis techniques to address these challenges remains an important direction for future work.

4.2.3 Efficiency & Resource Consumption. We also measure the efficiency and resource consumption of each schema routing method. Table 5 shows the number of queries per second (QPS), the duration of build time, the amount of disk space used, and the consumption of system and GPU memory. LLM inference and complex post-processing are the main reasons for the slowness

Table 5: Method efficiency & resource consumption.

Method	QPS[1]	Build (s)[2]	Disk (MB)	RAM (MB)	GPU (MB)
BM25	1677.1	10	36	800	-
SXFMR	49.0	48	420	2,230	3742
$\text{CRUSH}_{\text{BM25}}$[3]	0.9	10	36	1104	-
$\text{CRUSH}_{\text{SXFMR}}$[3]	0.8	48	420	2502	3742
BM25	1677.1	6972	36	800	-
DTR	49.0	5161	420	2,230	3742
DBCopilot	43.0	13678	852	656	11807

[1]Queries per second (QPS) and memory consumption is calculated with a query batch size of 64.
[2]Build time includes the time for model training and index construction.
[3]Computations are performed on the same device except for CRUSH, which requires commercial LLM inference.

of CRUSH approaches. DBCopilot, based on an encoder-decoder architecture, consumes more disk space and GPU memory than dense retrieval approaches that based on an encoder-only architecture. Despite the slightly higher resource consumption for training and inference in generative retrieval, we believe and have observed that the growing popularity of generative AI is leading to an increasing number of proposed optimizations.

4.3 Schema-Agnostic NL2SQL (RQ3 & RQ4)

This section compares the performance of schema-agnostic NL2SQL under various situations, including providing gold schemata, using different schema routing methods, and prompt strategies. Experimental results are shown in Table 6.

Oracle Test. We first conduct oracle tests to access the upper bound of schema-agnostic NL2SQL, as well as the impact of irrelevant schema elements for LLM-based NL2SQL, by providing gold schemata at various small scales. Specifically, we gradually move from 5 database schemata to the gold database, then to gold tables only, or even to gold tables with only gold columns. Experimental results in Table 6 show that as the number of schema elements increases, the EX of LLM-generated SQL queries continues to decrease significantly, while the inference cost keeps increasing many times over. This suggests that LLMs, like PLMs [37], are also affected by extraneous schema elements in SQL generation. Predictably, the situation gets worse as the schema scales. As a result, it is highly inefficient and ineffective to expose massive database schemata directly to LLMs, making schema routing necessary for schema-agnostic NL2SQL.

Finding 4. Schema routing is necessary for scaling natural language querying to massive databases.

Impact of Improved Recall. We then perform schema-agnostic NL2SQL to verify the impact of improved recall on the accuracy of generated SQL queries. Without loss of generality, we compare DBCopilot to sparse $\text{CRUSH}_{\text{BM25}}$ and dense DTR baselines, which achieve suboptimal routing performance on Spider and Bird, respectively. As shown in Table 6, DBCopilot outperforms $\text{CRUSH}_{\text{BM25}}$ and DTR by 4.43%~11.22% EX when using the best schema prompting strategy and by 0.85%~20.07% EX when employing human in the loop to select the optimal schema from candidates. We compare and discuss the prompting strategies in the next paragraph. The LLM inference cost of DBCopilot is slightly higher, possibly due to its successful retrieval of more required tables. The correlation between schema routing recall and SQL generation EX can also be confirmed by comparing these two baselines. $\text{CRUSH}_{\text{BM25}}$ outperforms DTR in recall and EX on Spider, while DTR achieves higher EX on the other two datasets due to its superior recall. Thus, the success of schema routing is essential for schema-agnostic NL2SQL.

Finding 5. DBCopilot leads in schema-agnostic NL2SQL due to its superior schema routing performance.

Exploration of Prompt Strategies. By routing multiple candidate schemata, the recall of the target schemata for NL questions can be greatly improved. We explore three prompt strategies for leveraging multiple candidate schemata during SQL generation. As shown in Table 6, when schema routing through DBCopilot, the performance of *multiple schema prompting* is slightly lower than that of *single schema prompting*. However, when schema routing through baselines, the performance of *multiple schema prompting* is typically better than that of *single schema prompting*. This is due to the trade-off between the improved recall brought by multiple candidate schemata and the impact of extraneous schemata on LLMs. As for *multiple schema COT prompting*, this strategy shows no absolute advantage over *multiple schema prompting*, and each wins and loses on different routing methods and datasets. Thus, the limitations of LLMs in distinguishing domain-specific schemata make it cost effective for DBCopilot to find the correct schema in the top 1 candidate. Finally, by introducing human-in-the-loop to select the desired schema from top 5 candidates, DBCopilot comes close to the ideal performance of providing the gold database schema, but with lower costs.

Finding 6. The best schema prompting strategy is optimal for DBCopilot unless there is human interaction.

Table 6: Evaluation of schema-agnostic NL2SQL with various gold schemata, different schema routing methods, and prompt strategies.

Routing Method	Prompt Template	Spider		Bird		$\text{Spider}_{\text{syn}}$	
		EX	Cost	EX	Cost	EX	Cost
		Oracle Test					
	Gold T. & C.	81.24	0.07	35.98	0.15	77.37	0.07
	Gold T.	76.60	0.09	31.03	0.24	68.67	0.09
	Gold DB	74.08	0.12	27.57	0.40	61.99	0.13
	5 DB w. Gold	70.21	0.13	18.38	8.80	51.64	0.40
		Best Schema Prompting					
$\text{CRUSH}_{\text{BM25}}$	Top 1	56.58	0.10	14.73	0.26	20.60	0.09
DTR	Top 1	43.91	0.11	19.04	0.33	26.21	0.10
DBCopilot	Top 1	**63.35**	0.11	**23.47**	0.36	**37.43**	0.11
		Multiple Schema Prompting					
$\text{CRUSH}_{\text{BM25}}$	Top 5	55.61	0.20	17.60	0.57	21.86	0.19
DTR	Top 5	54.45	0.27	19.56	0.79	32.59	0.27
DBCopilot	Top 5	61.51	0.31	21.45	0.95	35.49	0.31
		Multiple Schema COT Prompting					
$\text{CRUSH}_{\text{BM25}}$	Top 5	53.97	0.28	22.10	0.83	22.34	0.26
DTR	Top 5	50.68	0.36	22.43	1.08	28.53	0.37
DBCopilot	Top 5	51.26	0.40	22.75	1.24	31.24	0.39
		Human in the Loop					
$\text{CRUSH}_{\text{BM25}}$	Top 5	69.34	0.10	25.10	0.32	34.72	0.09
DTR	Top 5	66.92	0.11	25.42	0.37	47.78	0.11
DBCopilot	Top 5	72.05	0.11	26.27	0.37	55.42	0.11

"T." and "C." stand for table and column, respectively. We bold best results of schema-agnostic NL2SQL and underline the best with human in the loop to select the optimal candidate schema. For retrieval baselines, a candidate schema consists of the top candidate database and the retrieved tables that belong to the database.

Table 7: Performance loss in ablation studies.

	Spider				Bird			
	Database		Table		Database		Table	
	R@1	R@5	R@5	R@15	R@1	R@5	R@5	R@15
DBCopilot	85.01	96.42	91.63	97.51	88.92	97.13	85.83	94.59
w/ BS[1]	-3.20	-2.90	-4.09	-5.75	-3.39	-1.63	-6.43	-5.99
w/ OD[1,2]	-84.72	-82.40	-88.76	-79.55	-88.92	-82.27	-85.73	-88.81
w/ MD[1]	-0.48	+0.20	-0.83	-0.64	-3.85	-0.72	-3.39	-0.81
w/o CD[1]	-0.39	-0.19	-4.15	-2.51	+0.12	-0.45	-2.81	-1.93
w/o DB[1]	+0.48	-3.00	+0.64	-2.88	-0.52	-4.89	-3.69	-8.87

[1]"BS", "OD", "MD", "CD", and "DB" stand for basic serialization, original NL2SQL training data, mixed synthetic and original training data, constrained decoding, and diverse beam search.

[2]Note that DBCopilot trained on original NL2SQL training data fails to work because training and test sets do not share the same databases, and generative retrieval is limited to generalizing to unseen schemata. However, according to baselines fine-tuned on synthetic data in Table 3 and Table 4, as well as ablation results on other modules, we would like to emphasize that data synthesis is not the only effective part.

4.4 Ablation Study (RQ5)

We perform ablation studies on DBCopilot to validate the contribution of each module, as summarized in Table 7.

DFS Serialization. DFS serialization is essential for schema routers to learn the underlying relations of databases and tables. To demonstrate this, we replace the DFS serialization with the unordered basic serialization of tables, similar to DSI for multiple documents. As shown in Table 7, this results in a 1% to 6% reduction in recall. In addition, DFS serialization facilitates graph-based constrained decoding to reduce the search space, either by first generating a database and then generating its tables, or by further generating based on existing tables.

Data Synthesis. Generative retrieval enables relation-aware end-to-end schema routing, but is unable to generalize to unseen schemata during training. We can see that routers trained on instances (N, S) extracted from original NL2SQL datasets are almost completely incapable of routing on unseen test schemata. Data synthesis by sampling schemata and generating pseudo-questions can effectively inject semantic mappings to model parameters. Besides, mixing original and synthetic data for training does not lead to more accurate schema routing, possibly due to label imbalance. Figure 10 shows how the performance varies with the amount of synthetic data. Database and table recalls increase rapidly at first as more data is synthesized, but level off after a certain amount. How to improve the quality of synthetic data and reduce the need for its quantity may be a future direction.

Decoding Strategy. Decoding optimizations (*i.e.,* constrained decoding and group beam search) are beneficial for generative schema routing during inference. Specifically, constrained decoding is useful for generating accurate table identifiers, since removing it mainly affects table recalls, about 2% to 4%. Meanwhile, group beam search helps to generate multiple diverse schemata because database recall@5 decreases significantly without it.

Finding 7. DFS serialization, data synthesis, and decoding optimizations contribute in different ways.

5 RELATED WORK

5.1 Natural Language to SQL

NL2SQL is an essential part of natural language interfaces to databases, enabling non-experts to easily query structured data.

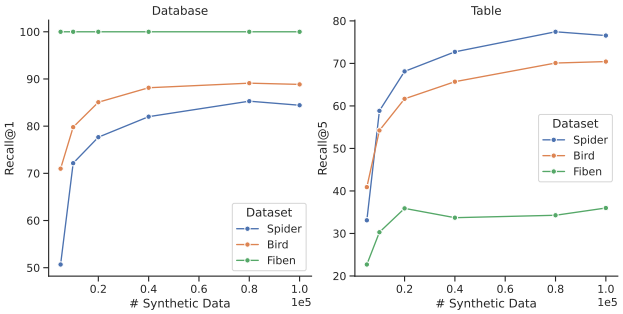

Figure 10: Database recall@1 and table recall@5 *w.r.t.* the number of (#) synthetic training data.

Most research has approached this task along the lines of semantic parsing [29, 42]. Early NL2SQL parsing systems adopted symbolic approaches that relied heavily on rule or grammar engineering, making them difficult to apply widely [78, 82]. In recent years, with the advances in deep learning, especially PLMs, NL2SQL approaches that using neural generation models have become the mainstream [24, 37, 39, 71, 80]. Some efforts have been made to improve encoding and decoding, such as relation-aware encoding [39, 71] and grammar-based decoding [61, 79]. Some other efforts have focused on generating accurate SQL queries through schema linking [37, 71] or skeleton generation [24, 25].

LLMs have emerged as a new paradigm for NL2SQL [22, 36, 45, 58]. Unlike previous work, a core component of the LLM-based NL2SQL solution is prompt engineering, *i.e.,* prompting LLM to generate correct and efficient SQL queries. Several efforts have been made to select appropriate in-context learning examples through question classification [55], skeleton retrieval [26], and in-domain example synthesis [7]. Research has also been done on decoupling zero-shot NL2SQL into PLM-based sketch generation and LLM-based SQL completion [25]. Recently, pipeline-based approaches decompose NL2SQL into steps including schema linking, query classification, query decomposition, SQL refinement, further extending the power of LLM-based NL2SQL [43, 55]. These studies are orthogonal to our work and can be easily integrated to optimize SQL generation of DBCopilot.

5.2 Table Discovery

Table discovery techniques are critical to data management, allowing users to explore and gain insights from data warehouses or lakes [17]. A common practice for table discovery is query-driven discovery [50, 84], where the query is typically in the form of keywords, tables, or NL questions. For keyword search, data discovery systems find relevant tables by searching for topic terms in the metadata [6, 54]. Table understanding [13], including domain discovery [41, 51] and table annotation [44, 68], has emerged as a means to mitigate the missing, incomplete, and inconsistent table schemata for keyword search. For table search, tables are retrieved to augment a given table as the query [60]. Joinable table search [14, 87] and unionable table search [18, 32] aim to augment a query table with additional attributes and new tuples, respectively. For NL question search, open domain question answering retrieves independent tables to answer the NL question with table-based models [9, 27]. Although dense retrieval methods based on contrastive learning have achieved advanced performance in the above tasks [14, 18, 27], they are

difficult to consider the structural relationships between tables and retrieve them jointly.

Alternatively, data navigation allows users to navigate interactively through the tables organized hierarchically [49] or relationally [19]. Our work can be seen as automated data navigation, where we employ relation-aware DSI to model the query-based navigation in an end-to-end manner. There is another line of research that focuses on table content discovery for Web table QA [3], which typically employs PLMs to manipulate and analyze HTML tables of limited size (with average #row < 20) [28]. However, these systems are not suitable for NLIDBs as they cannot efficiently handle large relational tables with thousands or millions of rows and diverse data types that are better handled through semantic parsing.

6 DISCUSSION & FUTURE WORK

DBCopilot has made significant progress in schema routing and schema-agnostic NL2SQL. However, there are some limitations that need to be addressed in future work.

1) **Cross-Databases Querying**. While current system shows the capability to route NL questions over massive databases, it is limited to generating SQL queries against a single database. Extending DBCopilot to support cross-database SQL querying faces several challenges: i) the heterogeneity of different databases and the complexity of ETL processes for cross-database SQL querying; ii) the increased difficulty in discovering implicit relationships between tables across multiple databases than within a single database; iii) the absence of comprehensive benchmarks for evaluating cross-database NL2SQL systems. We envision an alternative direction to decompose complex NL questions into multiple sub-questions, each answerable by a single database, and then integrate the results.

2) **Synthetic Data Quality**. As discussed in the case study, the quality of synthetic data used for training has a notable impact on router performance. Future efforts could focus on: i) generating high-quality synthetic data by using larger LLMs with better factual consistency; ii) providing more schema descriptions to reduce hallucination; iii) implementing automated validation mechanisms using TNLI [5] to identify and remove low-quality synthetic data. In addition, reducing the amount of data used to train schema routers while maintaining their performance is also a worthwhile exploration.

3) **Dynamic Schema Update**. In real-world scenarios, database schemata may evolve over time, which creates a need for all retrieval-based methods to update the index. Although we have decoupled the dynamic schema routing into a compact and flexible copilot model, and it currently takes less than 4 hours to train a schema router from scratch on thousands of tables, managing to incremental update DSI can further reduce the computational resources required for evolving schemata.

7 CONCLUSION

In this paper, we present DBCopilot, a framework for effectively scaling natural language querying to massive databases. DBCopilot decouples schema-agnostic NL2SQL into schema routing and SQL generation through LLM-Copilot collaboration. We present an approach to effectively navigate the target database and tables for an NL question in a relation-aware, end-to-end manner. We also propose a training data synthesis paradigm to adapt routers over massive databases without manual intervention, and explore various prompt strategies to incorporate multiple candidate schemata in LLM-based SQL generation. Extensive experiments verify the effectiveness of our solution. In summary, DBCopilot pushes the boundaries of NL2SQL, enabling researchers to better strategize their quest for data accessibility.

ACKNOWLEDGMENTS

We sincerely thank the reviewers for their insightful comments and valuable suggestions. This work was supported in part by Beijing Natural Science Foundation under Grant L243006, and in part by the Natural Science Foundation of China under Grant No. 62476265 and 62306303.

REFERENCES

[1] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. *VLDB J.* 28, 5 (2019), 793–819. https://doi.org/10.1007/s00778-019-00567-8

[2] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases - an introduction. *Nat. Lang. Eng.* 1, 1 (1995), 29–81. https://doi.org/10.1017/S135132490000005X

[3] Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for Tabular Data Representation: A Survey of Models and Applications. *Trans. Assoc. Comput. Linguistics* 11 (2023), 227–249. https://doi.org/10.1162/TACL_A_00544

[4] Lukas Berglund, Meg Tong, Maximilian Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2024. The Reversal Curse: LLMs trained on "A is B" fail to learn "B is A". In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=GPKTIktA0k

[5] Jean-Flavien Bussotti, Enzo Veltri, Donatello Santoro, and Paolo Papotti. 2023. Generation of Training Examples for Tabular Natural Language Inference. *Proc. ACM Manag. Data* 1, 4 (2023), 243:1–243:27. https://doi.org/10.1145/3626730

[6] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. 2009. Data Integration for the Relational Web. *Proc. VLDB Endow.* 2, 1 (2009), 1090–1101. https://doi.org/10.14778/1687627.1687750

[7] Shuaichen Chang and Eric Fosler-Lussier. 2023. Selective Demonstrations for Cross-domain Text-to-SQL. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 14174–14189. https://aclanthology.org/2023.findings-emnlp.944

[8] Peter Baile Chen, Yi Zhang, and Dan Roth. 2024. Is Table Retrieval a Solved Problem? Exploring Join-Aware Multi-Table Retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 2687–2699. https://doi.org/10.18653/V1/2024.ACL-LONG.148

[9] Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W. Cohen. 2021. Open Question Answering over Tables and Text. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=MmCRswl1UYl

[10] Xiaoyang Chen, Yanjiang Liu, Ben He, Le Sun, and Yingfei Sun. 2023. Understanding Differential Search Index for Text Retrieval. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 10701–10717. https://doi.org/10.18653/V1/2023.FINDINGS-ACL.681

[11] Defog.ai. 2023. Defog SQLCoder. https://github.com/defog-ai/sqlcoder

[12] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-Grounded Pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 1337–1350. https://doi.org/10.18653/v1/2021.naacl-main.105

[13] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proc. VLDB Endow.* 14, 3 (2020), 307–319. https://doi.org/10.5555/3430915.3442430

[14] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 456–467. https://doi.org/10.1109/ICDE51399.2021.00046

[15] Zhen Dong, Shizhao Sun, Hongzhi Liu, Jian-Guang Lou, and Dongmei Zhang. 2019. Data-Anonymous Encoding for Text-to-SQL Generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 5404–5413. https://doi.org/10.18653/V1/D19-1543

[16] William Falcon and The PyTorch Lightning team. 2019. *PyTorch Lightning*. https://doi.org/10.5281/zenodo.3828935

[17] Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. 2023. Table Discovery in Data Lakes: State-of-the-art and Future Directions. In *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, Sudipto Das, Ippokratis Pandis, K. Selçuk Candan, and Sihem Amer-Yahia (Eds.). ACM, 69–75. https://doi.org/10.1145/3555041.3589409

[18] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *Proc. VLDB Endow.* 16, 7 (2023), 1726–1739. https://www.vldb.org/pvldb/vol16/p1726-fan.pdf

[19] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 1001–1012. https://doi.org/10.1109/ICDE.2018.00094

[20] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. 2024. NL2SQL is a solved problem... Not!. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p74-floratou.pdf

[21] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021. Towards Robustness of Text-to-SQL Models against Synonym Substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 2505–2515. https://doi.org/10.18653/v1/2021.acl-long.195

[22] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (2024), 1132–1145. https://doi.org/10.14778/3641204.3641221

[23] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. *CoRR* abs/2312.10997 (2023). https://doi.org/10.48550/ARXIV.2312.10997 arXiv:2312.10997

[24] Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot Text-to-SQL Translation using Structure and Content Prompt Learning. *Proc. ACM Manag. Data* 1, 2 (2023), 147:1–147:28. https://doi.org/10.1145/3589292

[25] Zihui Gu, Ju Fan, Nan Tang, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Sam Madden, and Xiaoyong Du. 2023. Interleaving Pre-Trained Language Models and Large Language Models for Zero-Shot NL2SQL Generation. *CoRR* abs/2306.08891 (2023). https://doi.org/10.48550/arXiv.2306.08891 arXiv:2306.08891

[26] Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang. 2023. Prompting GPT-3.5 for Text-to-SQL with Desemanticization and Skeleton Retrieval. In *PRICAI 2023: Trends in Artificial Intelligence - 20th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2023, Jakarta, Indonesia, November 15-19, 2023, Proceedings, Part II (Lecture Notes in Computer Science)*, Fenrong Liu, Arun Anand Sadanandan, Duc Nghia Pham, Petrus Mursanto, and Dickson Lukose (Eds.), Vol. 14326. Springer, 262–274. https://doi.org/10.1007/978-981-99-7022-3_23

[27] Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Martin Eisenschlos. 2021. Open Domain Question Answering over Tables via Dense Retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 512–519. https://doi.org/10.18653/V1/2021.NAACL-MAIN.43

[28] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. *Proc. ACM Manag. Data* 1, 1 (2023), 30:1–30:17. https://doi.org/10.1145/3588710

[29] Aishwarya Kamath and Rajarshi Das. 2019. A Survey on Semantic Parsing. In *1st Conference on Automated Knowledge Base Construction, AKBC 2019, Amherst, MA, USA, May 20-22, 2019*. https://doi.org/10.24432/C5WC7D

[30] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6769–6781. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.550

[31] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *VLDB J.* 32, 4 (2023), 905–936. https://doi.org/10.1007/s00778-022-00776-8

[32] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proc. ACM Manag. Data* 1, 1 (2023), 9:1–9:25. https://doi.org/10.1145/3588689

[33] Mayank Kothyari, Dhruva Dhingra, Sunita Sarawagi, and Soumen Chakrabarti. 2023. CRUSH4SQL: Collective Retrieval Using Schema Hallucination For Text2SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 14054–14066. https://aclanthology.org/2023.emnlp-main.868

[34] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the Role of Schema Linking in Text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6943–6954. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.564

[35] Mosh Levy, Alon Jacoby, and Yoav Goldberg. 2024. Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 15339–15353. https://doi.org/10.18653/V1/2024.ACL-LONG.818

[36] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? [Experiment, Analysis \u0026 Benchmark]. *Proc. VLDB Endow.* 17, 11 (2024), 3318–3331. https://www.vldb.org/pvldb/vol17/p3318-luo.pdf

[37] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, Brian Williams, Yiling Chen, and Jennifer Neville (Eds.). AAAI Press, 13067–13075. https://doi.org/10.1609/aaai.v37i11.26535

[38] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proc. ACM Manag. Data* 2, 3 (2024), 127. https://doi.org/10.1145/3654930

[39] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023. Graphix-T5: Mixing Pre-trained Transformers with Graph-Aware Layers for Text-to-SQL Parsing. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, Brian Williams, Yiling Chen, and Jennifer Neville (Eds.). AAAI Press, 13076–13084. https://doi.org/10.1609/aaai.v37i11.26536

[40] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html

[41] Keqian Li, Yeye He, and Kris Ganjam. 2017. Discovering Enterprise Concepts Using Spreadsheet Tables. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 1873–1882. https://doi.org/10.1145/3097983.3098102

[42] Zhuang Li, Lizhen Qu, and Gholamreza Haffari. 2020. Context Dependent Semantic Parsing: A Survey. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, Donia Scott, Núria Bel, and Chengqing Zong (Eds.). International Committee on Computational Linguistics, 2509–2521. https://doi.org/10.18653/v1/2020.coling-main.226

[43] Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024. PET-SQL: A Prompt-enhanced Two-stage Text-to-SQL Framework with Cross-consistency. *CoRR* abs/2403.09732 (2024). https://doi.org/10.48550/ARXIV.2403.09732 arXiv:2403.09732

[44] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proc. VLDB Endow.* 3, 1 (2010), 1338–1347. https://doi.org/10.14778/1920841.1921005

[45] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. 2023. A comprehensive evaluation of ChatGPT's zero-shot Text-to-SQL capability. *CoRR* abs/2303.13547 (2023). https://doi.org/10.48550/arXiv.2303.13547 arXiv:2303.13547

[46] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Trans. Assoc. Comput. Linguistics* 12 (2024), 157–173. https://doi.org/10.1162/TACL_A_00638

[47] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=Bkg6RiCqY7

[48] Toby Mao. 2023. Python SQL Parser and Transpiler. https://github.com/tobymao/sqlglot

[49] Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J. Miller. 2020. Organizing Data Lakes for Navigation. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1939–1950. https://doi.org/10.1145/3318464.3380605

[50] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *Proc. VLDB Endow.* 12, 12 (2019), 1986–1989. https://doi.org/10.14778/3352063.3352116

[51] Masayo Ota, Heiko Mueller, Juliana Freire, and Divesh Srivastava. 2020. Data-Driven Domain Discovery for Structured Datasets. *Proc. VLDB Endow.* 13, 7 (2020), 953–965. https://doi.org/10.14778/3384345.3384346

[52] Simone Papicchio, Paolo Papotti, and Luca Cagliero. 2023. QATCH: Benchmarking SQL-centric tasks with Table Representation Learning Models on Your Data. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/62a24b69b820d30e9e5ad4f15ff7bf72-Abstract-Datasets_and_Benchmarks.html

[53] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph Retrieval-Augmented Generation: A Survey. *CoRR* abs/2408.08921 (2024). https://doi.org/10.48550/ARXIV.2408.08921 arXiv:2408.08921

[54] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web using Column Keywords. *Proc. VLDB Endow.* 5, 10 (2012), 908–919. https://doi.org/10.14778/2336664.2336665

[55] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/72223cc66f63ca1aa59edaec1b3670e6-Abstract-Conference.html

[56] William Pride. 2023. Text to SQL in production. https://canvasapp.com/blog/text-to-sql-in-production [Accessed 28-12-2023].

[57] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. http://jmlr.org/papers/v21/20-074.html

[58] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the Text-to-SQL Capabilities of Large Language Models. *CoRR* abs/2204.00498 (2022). https://doi.org/10.48550/arXiv.2204.00498 arXiv:2204.00498

[59] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, 3980–3990. https://doi.org/10.18653/v1/D19-1410

[60] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 817–828. https://doi.org/10.1145/2213836.2213962

[61] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 9895–9901. https://doi.org/10.18653/v1/2021.emnlp-main.779

[62] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish R. Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. ATHENA++: Natural Language Querying for Complex Nested SQL Queries. *Proc. VLDB Endow.* 13, 11 (2020), 2747–2759. http://www.vldb.org/pvldb/vol13/p2747-sen.pdf

[63] Numbers Station. 2023. Numbers Station Text to SQL model. https://github.com/NumbersStationAI/NSQL

[64] David Stuart. 2015. The Data Revolution: Big Data, Open Data, Data Infrastructures and Their Consequences. *Online Inf. Rev.* 39, 2 (2015), 272. https://doi.org/10.1108/OIR-01-2015-0011

[65] Weiwei Sun, Lingyong Yan, Zheng Chen, Shuaiqiang Wang, Haichao Zhu, Pengjie Ren, Zhumin Chen, Dawei Yin, Maarten de Rijke, and Zhaochun Ren. 2023. Learning to Tokenize for Generative Retrieval. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/91228b942a4528cdae031c1b68b127e8-Abstract-Conference.html

[66] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *CoRR* abs/2405.16755 (2024). https://doi.org/10.48550/ARXIV.2405.16755 arXiv:2405.16755

[67] Yi Tay, Vinh Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Prakash Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer Memory as a Differentiable Search Index. In *NeurIPS*. http://papers.nips.cc/paper_files/paper/2022/hash/892840a6123b5ec99ebaab8be1530fba-Abstract-Conference.html

[68] Petros Venetis, Alon Y. Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering Semantics of Tables

on the Web. *Proc. VLDB Endow.* 4, 9 (2011), 528–538. https://doi.org/10.14778/2002938.2002939

[69] Ashwin K. Vijayakumar, Michael Cogswell, Ramprasaath R. Selvaraju, Qing Sun, Stefan Lee, David J. Crandall, and Dhruv Batra. 2016. Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models. *CoRR* abs/1610.02424 (2016). arXiv:1610.02424 http://arxiv.org/abs/1610.02424

[70] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (Eds.). Association for Computational Linguistics, 540–557. https://aclanthology.org/2025.coling-main.36/

[71] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

[72] Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, Xing Xie, Hao Sun, Weiwei Deng, Qi Zhang, and Mao Yang. 2022. A Neural Corpus Indexer for Document Retrieval. In *NeurIPS*. http://papers.nips.cc/paper_files/paper/2022/hash/a46156bd3579c3b268108ea6aca71d13-Abstract-Conference.html

[73] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. *Trans. Mach. Learn. Res.* 2022 (2022). https://openreview.net/forum?id=yzkSU5zdwD

[74] Wikipedia contributors. 2023. Logical schema — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Logical_schema&oldid=1152291307 [Online; accessed 13-May-2024].

[75] Wikipedia contributors. 2023. Okapi BM25 — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Okapi_BM25&oldid=1180365145 [Online; accessed 16-October-2023].

[76] Wikipedia contributors. 2023. Path (graph theory) — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Path_(graph_theory)&oldid=1167548844 [Online; accessed 10-October-2023].

[77] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, Qun Liu and David Schlangen (Eds.). Association for Computational Linguistics, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

[78] William A. Woods. 1973. Progress in natural language understanding: an application to lunar geology. In *American Federation of Information Processing Societies: 1973 National Computer Conference, 4-8 June 1973, New York, NY, USA (AFIPS Conference Proceedings)*, Vol. 42. AFIPS Press/ACM, 441–450. https://doi.org/10.1145/1499586.1499695

[79] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. *CoRR* abs/1711.04436 (2017). arXiv:1711.04436 http://arxiv.org/abs/1711.04436

[80] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2021. GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=kyaIeYj4zZ

[81] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, 3911–3921. https://doi.org/10.18653/v1/d18-1425

[82] John M. Zelle and Raymond J. Mooney. 1996. Learning to Parse Database Queries Using Inductive Logic Programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2*, William J. Clancey and Daniel S. Weld (Eds.). AAAI Press / The MIT Press, 1050–1055. http://www.aaai.org/Library/AAAI/1996/aaai96-156.php

[83] Peitian Zhang, Zheng Liu, Yujia Zhou, Zhicheng Dou, and Zhao Cao. 2023. Term-Sets Can Be Strong Document Identifiers For Auto-Regressive Search Engines. *CoRR* abs/2305.13859 (2023). https://doi.org/10.48550/arXiv.2305.13859 arXiv:2305.13859

[84] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1951–1966. https://doi.org/10.1145/3318464.3389726

[85] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-Augmented Generation for AI-Generated Content: A Survey. *CoRR* abs/2402.19473 (2024). https://doi.org/10.48550/ARXIV.2402.19473 arXiv:2402.19473

[86] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. *CoRR* abs/2303.18223 (2023). https://doi.org/10.48550/arXiv.2303.18223 arXiv:2303.18223

[87] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 847–864. https://doi.org/10.1145/3299869.3300065