UNIVERSITY OF AMSTERDAM
Economics & Business

A Lean Retrieval-Augmented LLM Pipeline for Enterprise-Scale Text-to-SQL

Cadei Paolo

Student Number: 14060876

Date of final version: July 9, 2025

Master: Data Science and Business Analytics

Specialisation: Business Analytics

Supervisor: Dr C.M. Claudia Orellana Rodriguez

Second Reader: Prof. dr. S.I. Ilker Birbil

Table of Contents

1 Statement of Originality

This document is written by Student Cadei Paolo who declares to take full responsibility for the contents of this document. I declare that the text and the work presented in this document is original and that no sources other than those mentioned in the text and its references have been used in creating it. I have not used generative AI (such as ChatGPT) to generate or rewrite text. UvA Economics and Business is responsible solely for the supervision of completion of the work and submission, not for the contents.

2 Abstract

Tabular data remains central to modern analytics (Fang et al., 2024), yet large, complex databases with thousands of columns and diverse SQL dialects challenge text-to-SQL systems. On the new enterprise benchmark Spider 2.0-Snow, the top execution accuracy drops to just 38.21% (Lei et al., n.d.).

This thesis aims to (i) design a lean, LLM-driven pipeline for Spider 2.0-Snow that reduces API usage, emissions and costs, (ii) restores or approaches state-of-the-art accuracy using lighter OpenAI models, and (iii) grounds the workflow in realistic agentic interactions with live Snowflake databases.

To that end, I propose a Retrieval-Augmented Generation (RAG) architecture with schema compression and augmentation, hybrid vector retrieval, and a dual-agent loop. On a 74% Spider 2.0-Snow split, the pipeline achieves 33.24% execution accuracy, using ≤6 LLM calls and ≤3 database queries per task, about half the token budget of competing systems, thus balancing efficiency and performance.

3 Introduction

3.1 Background

Tabular data remains one of the fundamental data formats (Fang et al., 2024), yet it poses its own challenges, from the variety of query languages to the ever-growing scale and heterogeneity of the data itself. Thanks to recent breakthroughs in generative AI (Jin et al., 2023), consider the rapid emergence and impressive capabilities of models such as Gemini, OpenAI o3, and Anthropic's Claudem, we can now leverage large language models to translate natural-language questions into database queries, a task known as text-to-SQL. Because contemporary LLMs are trained on vast and diverse corpora, they can adapt those queries to multiple SQL dialects (Snowflake, BigQuery, SQLite, etc.) and even SQL-based frameworks such as dbt. This capability is essential for everyday querying, coordinating data workflows, and executing sophisticated business-intelligence analyses (Lei et al., 2024, p. 1).

A plethora of scholars confirm that LLMs excel at code-generation tasks, including text-to-SQL (Chen et al., 2021; Austin et al., 2021). Pipelines built on GPT-4, for example, have achieved very high execution accuracy on the classic benchmarks: 91.2% on Spider 1.0 (Yu et al., 2018) and 73.0% on BIRD (Li et al., 2024). Yet these datasets remain simplistic compared with real-world scenarios, where SQL dialects vary, schemas reach thousands of columns, data is messy, and external documentation is often required (Lei et al., 2024, p. 1).

To close that realism gap, Lei et al. (2024, p. 1), the authors of Spider 2.0, introduced a new benchmark explicitly designed to mirror enterprise-scale complexity. Spider 2.0 features schemas exceeding 1000 columns, nested structures, and the need to consult documentation, metadata, and even code bases while

generating answers. It spans several SQL dialects, Snowflake, BigQuery, DuckDB, SQLite, and dbt, and it demands far longer, more intricate SQL outputs. Finally, the benchmark is set in an agentic context: a model must search, reason, and interact with multiple information sources throughout the workflow, bringing the task much closer to production-grade text-to-SQL challenges.

With the advent of Spider 2.0, existing text-to-SQL methods now record strikingly low execution-accuracy scores. The current best result on Spider 2.0-Snow is 38.21%, while earlier state-of-the-art systems that excelled on simpler benchmarks, DAIL-SQL + GPT-4o, CHESS + GPT-4o, and DIN-SQL + GPT-4o, achieve only 2.20%, 1.28%, and 0.00%, respectively (Lei et al., n.d.). This sharp performance drop reflects code agents' difficulties in long-context settings: they can misinterpret instructions, miss key parts of the schema, or produce output that is inconsistent and unreliable (Xia et al., 2024). Moreover, previous frameworks lack robust support for multiple SQL dialects and for schemas enriched with complex metadata (Deng et al., 2025).

Below is a table showing a summary of the differences between Spider 2.0 and previous benchmark databases on different dimensions:

Dimension	**Spider 2.0**	**Earlier Benchmarks**
Database Size and Schema Complexity	Enterprise-scale schemas; often 1000+ columns, nested structures, and multiple schema per DB	Small to medium schemas; tipically <100 columns, flat structure
Context provided to the model	Question plus project code, documentation, metadata, and dialect references	Question and schema only
SQL dialect coverage	Multiple Dialects (Snowflake, BigQuery, DuckDB, SQLite, dbt, etc.)	Usually a single dialect
Query complexity	Multi-step, sometimes 100+ lines or multi-query workflows	Single, comparatively short queries
Advanced SQL features	Frequent use of enterprise functions (e.g., ST_DISTANCE, CORR, window functions)	Advanced functions rarely required
State-of-the-art execution accuracy	15 ~ 38% (Spider 2.0-Snow)	73 ~ 91% (Spider 1.0, BIRD)
Real-world / agentic setting	High: model must search, reason, and interact with diverse sources during generation	Moderate: limited interaction, simpler reasoning context

Table 1 Differences between Spider 2.0 and previous benchmarks

3.2 Research Question – Objective

The ultimate objective of this project is to design an LLM-driven pipeline that tackles the Spider 2.0-Snow text-to-SQL challenge. Specifically, the aim is to match, or closely approach, state-of-the-art accuracy while cutting costs, both in API usage and environmental impact, by minimising the number of model calls and by relying on lighter, lower-priced OpenAI models.

3.3 Outline of thesis chapters

The remainder of this thesis is organised as follows. First, the next chapter reviews prior work on text-to-SQL and schema understanding, breaking the literature down into its key sub-topics. Then, I present a comprehensive overview of the methodology, explaining each step and component of the proposed pipeline in detail. This is followed by the results chapter, which covers the dataset, evaluation metrics, model configurations used in my experiments, and the results of my experiments. Finally, the thesis draws from the previous sections to summarise the findings, outline the main limitations, and highlight potential directions for future research.

4 Literature Review

The upcoming literature review will discuss four main topics related to the use of LLMs on real-world, enterprise-level text-to-SQL tasks: schema preprocessing and representation, schema structural linking, particularly in the use of schema matching, text-to-SQL methods, and evaluation metrics.

The focus on these topics stems from their relevance to my project's objectives: transforming raw database schemas; exploring the newly available dataset Spider 2.0 (Lei et al., 2024) to further academic research on corporate-like datasets; and developing and evaluating a framework to leverage LLMs to answer natural language questions.

4.1 Schema Understanding

Schemas form the foundational structure of databases, defining how data elements relate to one another and establishing the framework for data organisation. The following section will focus on the work done throughout the years in the fields of schema preprocessing, presentation, linking and matching.

4.1.1 Schema Preprocessing and Representation

Schema preprocessing and schema representation are crucial for downstream tasks such as schema matching and linking. These steps have been found to have a significant effect on subsequent downstream matching operations (Boerrigter, 2021; Paton et al., 2023; Chronas, 2023; Qiang, Taylor & Wang, 2025). In the following section, I will provide a comprehensive literature of the different methods that can be used to preprocess and represent the data, ranging from Information Analysis to schema normalisation, aliasing and schema pruning.

4.1.2 Information Analysis

Early efforts considered the study of Information Analysis (IA), defined as creating a description of the data structure both unambiguous and in terms of BR (binary-relationships) (Shoval & Even-Chaime, 1987). IA played a significant but evolving role in database schema analysis as early experiments found that normalisation for schema design was superior in less complex tasks with less time investment and not significantly inferior for more complex tasks (Shoval & Even-Chaime, 1987). Therefore, the role of IA shifted to tasks less focused on schema understanding. These include quantifying schema redundancy and integrity trade-offs (e.g. 3NF (Kolahi & Libkin, 2008)), anticipating schema evolution using predictive methods to adjust initial designs for future requirements (Bounif, 2003), and assessing schema quality and the impacts of change through database schema evolution analysis, such as DAHLIA (Meurice & Cleve, 2014).

4.1.3 Schema Normalisation

Schema normalisation plays a key role in supporting these tasks and was identified early on as the most effective and preferred approach (Shoval & Even-Chaime, 1987). A distinction in the study of schema normalisation can be found in whether the proposed approaches assume that functional dependencies (FD) are given or that they have to be discovered.

The former studies include the Boyce-Codd normal form (BCNF) (Nambiar et al., 1996) which extended normalisation framework to SQL; the XML-to-Relational Schema Mapping (Ahmed & Oakasha, 2005) used to obtain an optimal mapping approach through normalisation; the use of "qualified FDs" to formalise constraints during schema integration (He & Ling, 2004) which can be used to normalise integrated schemas; and a method to map XML DTD to relational schemas (Ahmad, 2008).

Removing the assumption of a given FD, DiScala and Abadi (2016) leveraged prior research on functional dependencies, schema matching, and mappings from semi-structured to relational formats to enable the transformation of nested key-value data into relational schemas. Alternative approaches to this transformation task have also been proposed. These include a simpler algorithm known as ARGO (Chasseur, Li & Patel, 2013), and a method based on XML graph structures developed by Shanmugasundaram et al. (2008). The issues with the approach DiScala and Abadi (2016) took is that their resulting schema, while similar to a normalised database schema, is not compliant with BCNF, 3NF, or 4NF. NORMALISE (Papenbrock et al. 2017) solves the issue ensuring BNCF while normalising relational datasets through discovered functional dependencies.

4.1.4 Schema Pruning and Aliasing

Schema aliasing can be used in many different contexts, ranging from schema preprocessing to schema migration. My focus will mainly concern its use in tandem with schema pruning in text-to-SQL tasks, as no studies on schema aliasing as a standalone technique have been found as of the writing of this paper. The only explicit mention of schema aliasing comes from Kyle Luoma and Arun Kumar (2025), who showed how the modification of existing schemas by making identifiers more natural improves the work between LLMs and databases. However, while schema aliasing has been shown to improve performance in text-to-SQL tasks (Luoma & Kumar, 2025), it is not a widely studied topic. On the other hand, schema pruning is a technique which is used in many text-to-SQL frameworks to reduce the impact of long-context windows which dilute the attention or exceed the input limitation of LLMs. Pruning, in practice, is necessary to deal with the amount of false positives in required columns recall (Maamari et al., 2024). Frameworks that use pruning techniques include: RSL-SQL (Cao et al. 2024), using forward and backward pruning, WAGG (Li et al., 2022) and ASTReS (Chen et al., 2024), using dynamical pruning, BRIDGE (Lin et al., 2020), using space-pruning heuristics, multilingual schema

pruning (Archanjo et al.,2023), PICARD (Scholak et al., 2021), using token pruning, and layer pruning (Sajjad et al., 2022).

4.1.5 Linking

In the following section I will consider the concepts of structural linking and schema linking as defined by Chen and colleagues (2020). Additionally, I will include schema matching, which complements these approaches by identifying correspondences between elements across different schemas, therefore facilitating effective linking at both the instance (schema) and structural levels. This makes it possible to link schema elements effectively, especially in cases where explicit constraints like foreign keys are missing.

4.1.6 Schema Linking and Structural Linking

Schema linking is the process of connecting elements in a natural language query (such as column names, table names, or values) to the corresponding elements in the database schema. It plays a crucial role in tasks like semantic parsing, question answering, and data exploration. Lei et al. (2020) found schema linking to be the crux of text-to-SQL task. This concept has been studied in many domains (Lei et al., 2020), ranging from matching entities in knowledge bases (Fu et al., 2020; Wu et al., 2019; Rijhwani et al., 2019; Logeswaran et al., 2019; Zhou et al., 2019; Wang et al., 2020), and converting natural language into structured data (Ren et al., 2018; Rastogi et al., 2017), to collecting key information from users in dialogue system (Su & Yan, 2017; Herzig & Berant, 2018).

However, my focus will be on its use in text-to-SQL tasks where it has mainly been treated as a secondary preprocessing step through the use of simple heuristics (Lei et al., 2020). Dong et al. (2019) point out that most previous studies use deterministic approaches like string matching or embedding similarity, and they are amoung the first to consider schema linking as a separate problem to study (Lei et al., 2020), combining sequential tagging with a two-stage anonymisation model (Dong et al., 2019). Further studies include incorporating schema linking as a trainable component within the network (Bogin et al., 2019; Wang et al., 2019). This technique, being part of the retrieval phase in a text-to-SQL model, uses multiple techniques that vary depending on two main factors: their representation of the schema and the methods they use to perform linking based on that representation (Maamari et al., 2024). The schema can be represented either through natural language (Glass et al., 2025) or more recent studies use code-like structures to represent the schema (Gao et al., 2023), while the techniques used range from immediately applying schema-filtering (Dong, et al, 2023; Lee et al., 2024; Talaei et al., 2024), to using intermediate structures to determine pertinent tables and columns (Qu et al., 2024). Moreover, Maamari et al. (2024) show that it is possible to forgo completely the use of schema linking in certain situations where the schema fits in the LLM's context window as new models' reasoning capabilities increase, reducing the need to remove false positives. However, this is not always the case,

as demonstrated by Guo et al. (2019), Li et al. (2024), Talaei et al. (2024), and Floratou et al. (2024). Last but not least, Want et al. (2025) proposed LinkAlign, a scalable solution for schema linking in text-to-SQL tasks for real-world databases, with a particular focus on Spider 2.0 (Lei et al., 2024).

While extensive research has been done on schema linking, structural linking is a topic that has not seen much research. While previous papers related to structural linking exist, in particular when considering studies related to structural mismatch between natural language and programming languages (Kwiatkowski et al., 2013; Berant et al., 2014; Guo et al., 2019), the ones to first define the process were Chen et al. (2020), who proposed a dynamic linking mechanism to switch between schema and structural linking. The lack of studies in this area is also due to the way academic databases were structured and the fact that the link between tables and databases was already given.

4.1.7 Foreign-Primary Key Identification

In real-world databases and certain benchmarks such as Spider 2.0 (Lei et al., 2024), functional dependencies such as primary keys (PKs) and foreign keys (FKs) are not explicitly provided. Since structural linking often relies on these relationships to traverse and connect elements across tables, discovering them becomes an essential step before performing schema matching or linking.

Most efforts focused on FK discovery (Lopes et al., 2002; Memari et al., 2015; Rostin et al., 2009; Zhang et al., 2010; Chen et al., 2014; Marchi et al., 2009; Motl & Kordík, 2017), however all of them assumed that PKs were given. Furthermore, Rostin et al. (2009) proposed machine learning algorithms to detect single-column foreign keys. Zhang et al. (2010) furthered the research by discovering multi-column foreign keys as well. Moreover, Chen et al. (2014) combined pruning and heuristic features to discover foreign keys. On the other hand, minimal studies have been conducted on the discovery of PKs, in particular Papenbrock and Naumann (2017) focused on distinguishing true Primary Keys using the fact that Keys are simply particular cases of UCCs (Unique Column Combinations) and INDs (Inclusion Dependencies) (Abedjan Z. et al., 2015). Following Meurice et al.'s (2014) and Chen et al.'s (2014) frameworks, Jan Motl and Pavel Kordík (2017) formulated an integer linear programming (ILP) problem to identify primary and foreign keys. Furthermore, building on the previuos research, Jiang et al. (2019) proposed HoPF (Holistic Primary Key and Foreign Key Detection) which is concerned with finding the true PK and FK from the sets of UCCs and INDs.

4.1.8 Schema Matching

While schema matching operates across different schemas to find equivalent or related elements, schema linking focuses on connecting those schema elements to parts of an input query. In this sense, schema matching can provide prior knowledge or enhance schema and structural linking by suggesting likely connections based on similarity or usage patterns.

When reviewing the literature related to schema matching, it is important to consider two different aspects of it: firstly, the evolution of its proposed taxonomy, and secondly, the different methods that can be used in view of the taxonomy proposed by Rahm and Bernstein (2001) (see Figure 1).

A first effort to categorise the different schema matching techniques was made by Batini, Lenzerini, and Navathe (1986), which excluded machine learning and ontological techniques. More elaborate was the approach taken by Rahm and Bernstein (2001), dividing techniques analysing schema, data instances or both, including a method using machine learning techniques. However, their survey was on linguistic and constraint instance-based approaches, all using element-level methods (Roger Blake, 2007). This classification was then furthered by Shvaiko and Euzenat (2005) to include granularity and the specific inputs used by the matching techniques. Another further classification was proposed by Doan and Halevy (2005) with a split between rule-based and learning-based methods. Building on top of the work done by Doan and Halevy (2005), Roger Blake (2007) added to the previous split the category of ontology-based techniques.

The following section will discuss the different methods for schema matching, building on top of the work done by Alwan et al. (2017), dividing them mainly using the taxonomy defined by Rahm and Bernstein (2001) (see Figure 1).

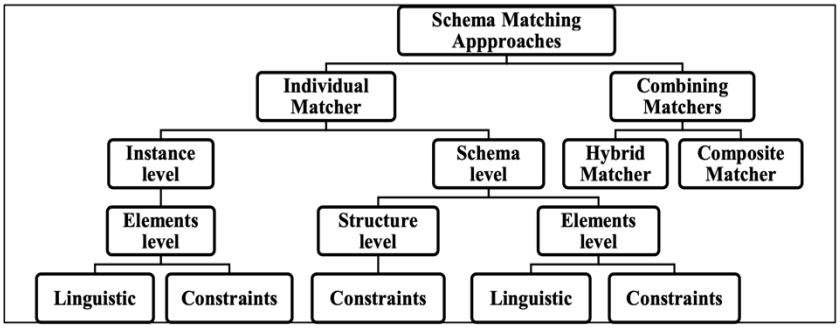

Figure 1 Classification of Schema Matching Methods

Firstly, the techniques that use instance-level matching include: N-gram (Leme et al., 2009; Mehdi et al., 2007), regular expressions (Zapilko et al., 2012), using syntactic techniques, and LSA (Latent Semantic Analysis) (Landauer et al., 1998), WordNet/Thesaurus (Fellbaum, 2010) and Google Similarity (Cilibrasi, 2007), using semantic methods (Alwan et al., 2017). Moreover, Alwan et al., (2017) found that the use of instance-based matching could be further divided into four different sub-parts. Firstly, into neural networks such as SEMINT (Li et al., 2000), SMDD (Alwan et al., 2017), CBSMA (Yang et al., 2008), as well as the approach proposed by Gozudeli et al. (2015), and OmniMatch (Koutras, 2020). Secondly, into machine learning techniques such as LSD (Rong et al., 2012), Autoplex (Berlin et al., 2002), Feng et al.'s (2009) method using attributes ranking and classification, as well as Mehdi et al.'s (2017) approach based on Google Similarity and Regular Expressions. Thirdly, into information theoretic proposed by Liang Y. (2008), Kang, J., and Naughton, J. F. (2003; 2008), and Wang et al. (2004), as well as the previously mentioned SMDD (Alwan et al., 2017). Lastly, constraint-based methods such as: Bilke & Naumann's (2005) duplicates-based model, C.E.H. Chua et al.'s model (2003)

using attribute identification, B. Zapilko et al.'s (2012) model using regular expressions, iMAP (Dhamankar et al., 2004), and Doan et al.'s CLIO (2003). Other instance-based approaches include the use of clustering (Zhang et al., 2011), an extension of the COMA method, namely COMA++ (Engmann, & Massmann, 2007), a linguistic-based approach, CLIO (Haas et al., 2005; Fagin et al., 2009), a hybrid-structure based approach, GLUE (Doan et al., 2003), an OWL approach, and EmdDI (Cappuzzo et al., 2020).

Secondly, the methods that use schema-level matching include DELTA (Benkley et al., 1995), DIKE (Palopoli et al., 2003), MOMIS (Bergamaschi et al., 1999), ARTEMIS (Giunchiglia et al., 2005), OPENII (Seligman et al., 2010), Similarity Flooding (Melnik et al., 2002), S-Match (Giunchiglia et al., 2004), CUPID (Madhavan et al., 2001), CLIO (Miller et al., 2001), SMAT (Zhang et al., 2021), ReMatch (Sheetrit et al., 2024), Matchmaker (Seedat, & van der Schaar, 2024), ADnEV (Shraga, 2020), pre-trained language models (Li et al., 2020), large language models (Zhang et al., 2023), and Jellyfish (Zhang et al., 2023).

Thirdly, hybrid and composite methods include SEMINT (Li et al., 2000), COMA++ (Engmann, & Massmann, 2007), AgreementMaker (Cruz et al., 2009), Data Tamer (Stonebraker et al., 2013), the hybrid model proposed by Sutanta et al. (2016), R. C. Fernandez et al.'s model (2018) using word embeddings, SemProp (Fernandez et al., 2018), OmniMatch (Koutras C., 2020), AiMatch (Hättasch et al., 2022), and SAHM (Asif-Ur-Rahman et al., 2023).

Additionally, research has been done on the use of graphs for schema matching, for instance Similarity Flooding (Melnik et al., 2002), S-Match (Giunchiglia et al., 2004), Aurum (Fernandez et al., 2018), OmniMatch and REMA (Koutras, 2020), KG-RAG4SM (Ma et al., 2025), and AbsMatcher (Engle et al., n.d.).

4.2 Text-to-SQL

Text-to-SQL (Text2SQL) or NL2SQL refers to the task of converting natural language queries into formal SQL statements that can be executed on relational databases. As a subfield of table understanding (Fang et al., 2024, p. 21), Text2SQL facilitates interaction with databases through natural language, making data extraction more accessible to non-expert users (Sun et al., 2024). This approach has significantly lowered the barriers to querying structured data (Zhang et al., 2020), while also broadening the range of applications in which relational databases can be effectively used (Sun et al., 2024, p. 2). In particular, the use of Text-to-SQL as an interface for real-world databases has increased, as it supports the application on unknown schema stuctures, working both across databases and independently from the schema it was trained on (Glass et al., 2025).

Three periods can be distinguished in the development of Text-to-SQL: the traditional stage (Zhang et al., 2020, p. 9) and the pre- and post-LLM stages (Sun et al., 2024).

Before the advent of LLMs, two primary stages can be identified: a traditional phase and a deep learning phase. The former includes rule-based and template-based approaches (Androutsopoulos, Ritchie & Thanisch, 1995; Hristidis, Papakonstantinou & Gravano, 2003; Li & Jagadish, 2014; Popescu, Etzioni & Kautz, 2003; Yaghmazadeh et al., 2017; Grosz, 1983), such as SQLizer (Yaghmazadeh et al., 2017) and TEAM (Grosz, 1983) respectively (Zhang et al., 2020). While the deep learning phase, includes sequence-to-sequence models like Seq2SQL (Zhong et al., 2017), SQL-Net (Xu et al., 2017), and RASAT (Qi et al., 2022), where the former two models used reinforcement learning techniques (Pourreza et al., 2025); transformer models like T5 (Raffel et al., 2023); derivative models focused on schema linking and encoding (Sun et al., 2024), graph encoders, and constraint generation, such as IRNET (Guo et al., 2019), and RAT-SQL (Wang et al., 2021); incremental parser models such as INCSQL (Shi et al., 2018), and PICARD (Scholak, Schucher & Bahdanau, 2021); and Graph Neural Networks (GNNs) like Graphix-T5 (Li et al., 2023).

The expanded input capacity of LLM models has enabled the inclusion of full database schemas, at least for small databases, directly within the prompt (Maamari et al., 2024). However, this approach presents several drawbacks mainly linked to impracticability for large databases and tables, which reduced LLMs accuracy due to long context, and increased costs (Fang et al., 2024, p. 9).

To address these limitations, researchers have developed various strategies, such as prompt engineering, fine-tuning, schema linkage, and retrieval-based strategies (Sun et al., 2024). Prompting techniques have evolved significantly, moving from general strategies, such as Chain-of-Thought (Wei et al., 2023) and Program-of-Thought (Chen et al., 2023), to more specialised approaches tailored for text-to-SQL. These include models like DIN-SQL (Pourreza & Rafiei, 2023), DAIL-SQL (Dao et al., 2023), ACT-SQL (Zhang et al., 2023), SuperSQL (Li et al., 2024), and ExCoT (Zhai et al., 2025), which leverages Chain-of-Thought (CoT) and Direct Preference Optimisation (DOP).

Although fine-tuning has shown promise, it remained until recently less studied to to its high costs. Nonetheless, recent contributions in this area include CodeS (Li et al., 2024), Domain-aware Fine-tuning for TableGPT (Zha et al., 2023), and reinforcement-learning fine-tuning (Nguyen et al., 2025; Pourrezza et al., 2025). Zha et al. (2023, pp. 2-3) went further by exploring alternatives such as Chain-of-Command prompting and embedding-based tabular data serialization technique called Global Table Representation.

Schema linking methods, such as RAT-SQL (Wang et al., 2021) and RESDSQL (Li et al., 2023), are typically not considered as standalone solutions but as supporting tools to guide other methods (Sun et al., 2024) (see Section 4.1.6 on Schema Linking).

Among newer developments, retrieval-based methods have shown promise by leveraging historical query data and advanced prompt-design strategies (Sun et al., 2024, pp. 5-6). For instance, SuperSQL (Li et al., 2024) analysed the use of different prompting techniques. Volvovsky, Marcass, and

Panbiharwala (2024) introduced DFIN-SQL, inspired by DIN-SQL (Pourreza & Rafiei, 2023) and utilising both prompt techniques and retrieval-augmented generation. Similarly, Talaei et al. proposed CHESS, which used contextual harnessing techniques.

Additionally, Thorpe, Duberstein, and Kinsey (2024, p. 1) improved SOTA, proposing two distinct models: Dubo-SQL v1, a fine-tuned GPT-3.5 Turbo, and Dubo-SQL v2, a retrieval-augmented version of GPT-4 Turbo (Thorpe, Duberstein & Kinsey, 2024, pp. 2-4).

Furthermore, more recent studies focuses on detecting and correcting erroneous SQL queries. For instance, SQLFixAgent (Cen et al., 2025), RSL-SQL (Cao et al., 2024), REFORCE (Deng et al., 2025), MAGESQL (Shen et al., 2025), which also include graph-based selection methods, and execution-guided SQL generation (Borchmann & Wydmuch, 2025) all apply various techniques to refine and correct the SQL query outputted by the LLM.

Another overview is presented by Figure 2 below, taken from the survey by Hong et al. (2025).

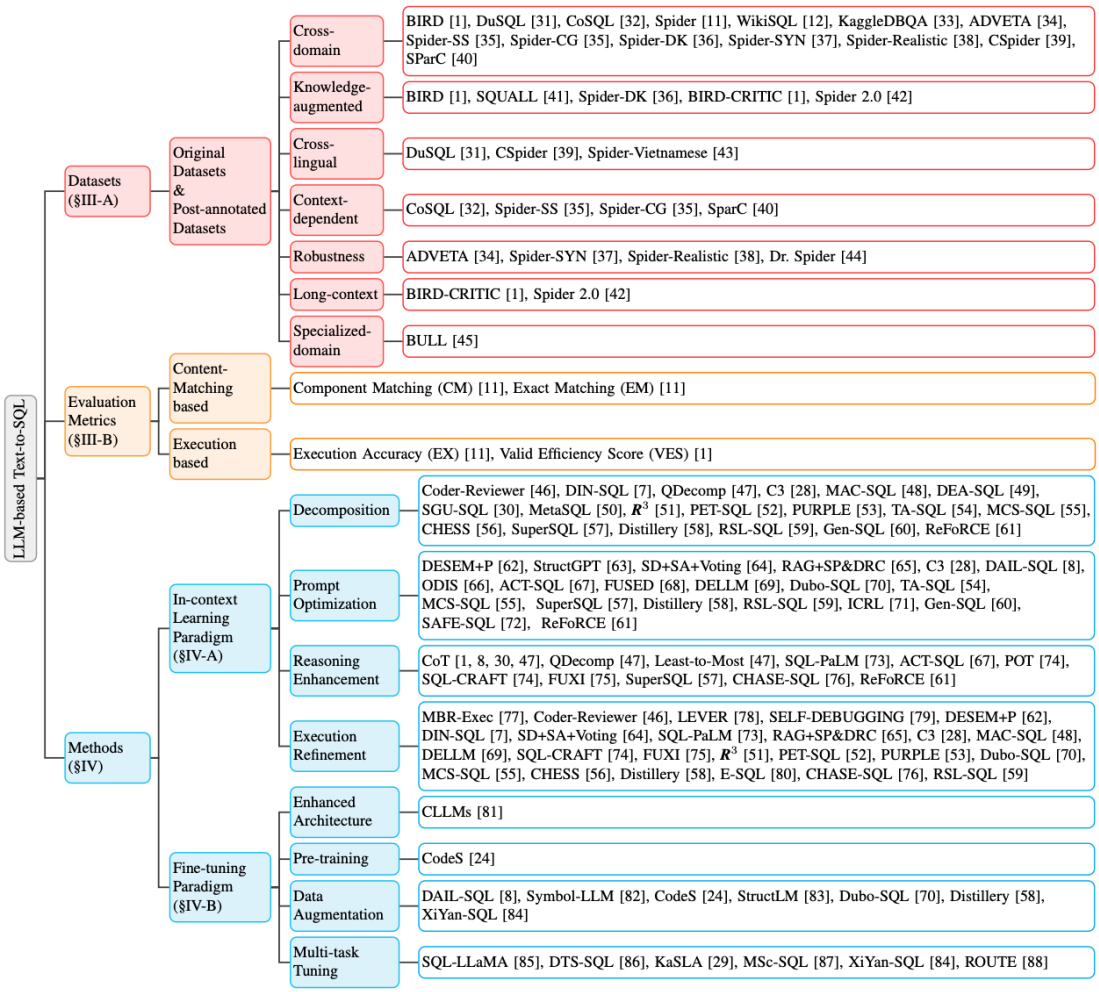

Figure 2 Taxonomy tree of research in LLM-based text-to-SQL (Hong et al., 2025)

5 Methodology

The methodology outlined in the following sections details the logic behind the Retrieval-Augmented Generation (RAG) pipeline developed to address the challenges introduced by the Spider 2.0-Snow evaluation framework. As shown in Figure 3, the pipeline is built to handle realistic text-to-SQL tasks that involve complex reasoning, long input contexts, and large-scale databases, in particular those hosted on Snowflake.

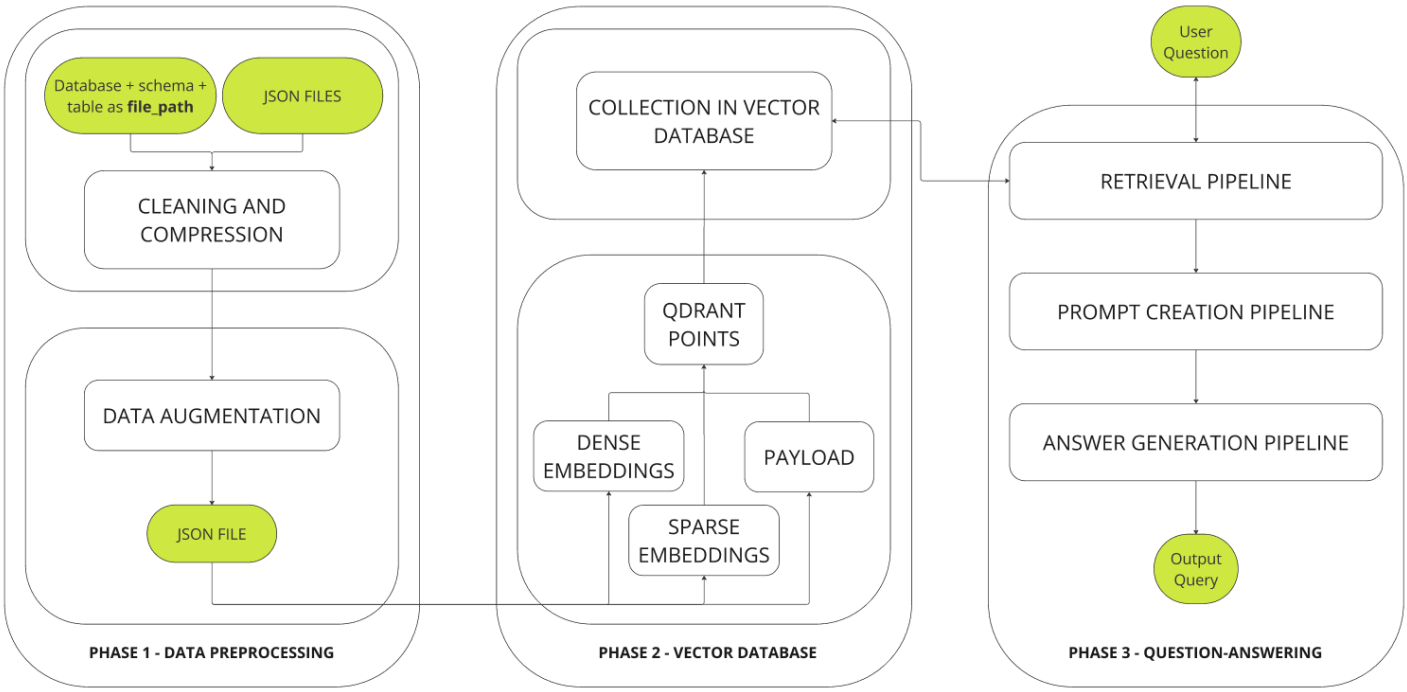

Figure 3 Pipeline-Steps Overview

The Data Preprocessing phase initiates the framework. This phase includes two separate steps: cleaning and compression of tables and schema related information, and data augmentation, where an LLM is used to infer descriptions for columns and tables. This two-step operation will be described in Section 5.1.

Subsequently, the pipeline continues to the vector database construction phase, which embeds and stores the information outputted by the previous step. This step includes the creation of dense and sparse embeddings, as well as the payload, which are afterwards uploaded as points to a collection in Qdrant (Qdrant, n.d.). This process will be presented in more detail in Section 5.2.

After the vector database has been created, the pipeline will use a three-steps process to answer the posed question by outputting a Snowflake SQL query. Firstly, using the question, I retrieve and rerank the relevant points in the vector database, which represent the columns in the tables of the database. Then, based on the retrieved columns and the consistency check done based on the Snowflake database,

a prompt to start the Answer Generation pipeline is created. Lastly, through a conversation between two LLMs, an answer is generated and saved. Section 5.3 elaborates further on all the steps of the Question-Answering Phase.

In the following Sections, I will explain in more detail all the parts outlined above, providing explanations related to the methods behind their implementation as well as the rationale behind these techniques.

5.1 Data Preprocessing

The Data Preprocessing phase concentrates on cleaning, compressing, and augmenting the data related to the databases found in the Spider 2.0-Snow dataset as much as possible. This phase is divided into two main steps (see Figure 4): firstly, cleaning and compression of tables, and secondly, data augmentation.

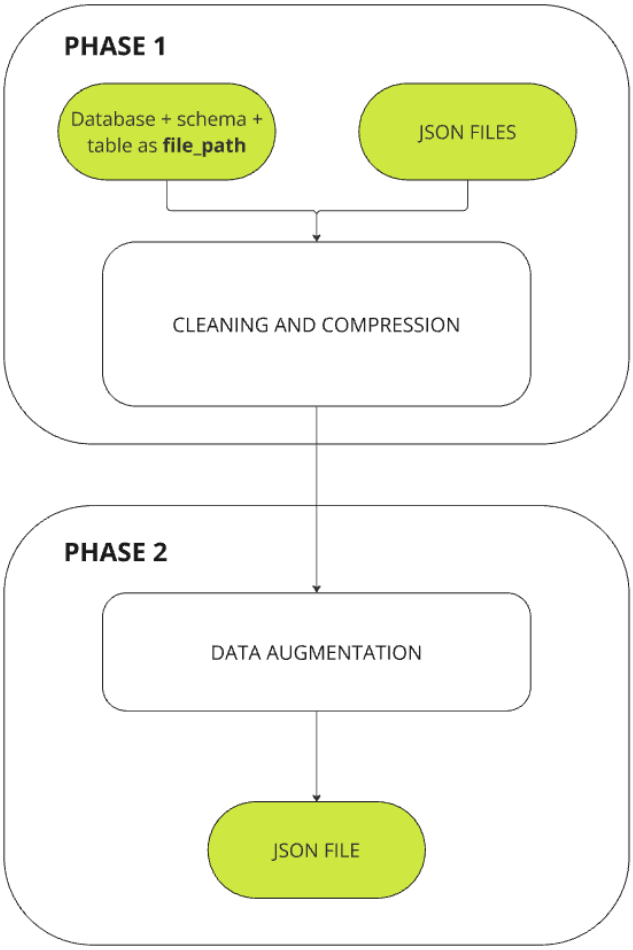

Figure 4 Data Preprocessing Steps Overview

5.1.1 Data Cleaning and Compression

a Input

The input of this phase consists of two elements: a list of paths, structured as database/schema/table, which point to the JSON files provided by the authors of the Spider 2.0 dataset; and the JSON files themselves, which include metadata such as table names, column names, column types, column descriptions (which may contain null values), and sample rows.

b Output

The output of this phase is a unified JSON file that collects all available metadata for the tables. Since tables have been compressed in this process (see Section 5.1.1c), the output file adopts two different structures, one for grouped tables and another for standalone tables (see Figure 5).

- Database (1st folder)
  - Schema (2nd folder)
    - **Grouped**
      - *Table Template* (.json)
        - description: table-level description generated by the LLM.
        - keywords: list of 10 keywords related to the table.
        - Lists of dictionaries sharing the same template, with:
          - variables: dictionary where keys are variable0, variable1, etc., and values are the variable options.
          - combinations: possible combinations of variables used in the template.
          - variable_order: sequence in which variables are placed into the template.
          - details: dictionary containing:
            - columns: column names and types.
            - description: column-level descriptions.
            - sample_row: sample row if provided.
            - keywords: 10 keywords per column.
          - prompt: the prompt used to request the LLM to generate the table-level description.
    - **Ungrouped**
      - *Table Name* (.json)
        - description: table-level description generated by the LLM.
        - keywords: 10 keywords related to the table.
        - details: dictionary containing:
          - columns: column names and types.
          - description: column-level descriptions.
          - sample_row: a sample row, if available.
          - prompt: prompt used to request the column-level descriptions and keywords.
          - keywords: dictionary where keys are column names and values are lists of 10 keywords.
        - prompt: prompt used for the table-level description and keywords.

Figure 5 JSON file structure of Data Preprocessing Phase Output

c Process

Although the dataset authors provide metadata, it still requires cleaning and restructuring, mainly due to the table grouping process. This grouping is essential because LLMs have limited context windows and perform worse with longer prompts (Levy, Jacoby & Goldberg, 2024). Many tables in the dataset are structurally identical but differ only in attributes like year, time interval (eg., 3-year vs. 5-year), or genomic reference.

To reduce redundancy, I grouped such tables under a shared template with variables representing differences. For example, in NOAA_DATA_PLUS/NOAA_GSOD,

```
{
  "DATABASE_NAME_1": {
    "SCHEMA_NAME": {
      "grouped": {
        "TEMPLATE_1": [
          {
            "combinations": [ ["VAR1_VAL", "VAR2_VAL", ...] ],
            "variable_order": [ "VAR1", "VAR2", ... ],
            "details": {
              "columns": {
                "column_name_1": "type_1",
                "column_name_2": "type_2",
                ...
              },
              "description": "some description from the file",
              "sample_row": [ [val1, val2, ...], [val1, val2, ...]
]
            }
          },
          ...
        ]
      },

      "ungrouped": {
        "TABLE_NAME": {
          "columns": {
            "column_name_1": "type_1",
            "column_name_2": "type_2",
            ...
          },
          "description": "some description from the file",
          "sample_row": [ [val1, val2, ...], [val1, val2, ...]
        }
      }
```

Figure 6 JSON file pseudocode of Data Preprocessing Phase Output

tables spanning 1929 to 2015 were merged into a single template, GSOD{variable0}, with variables [1929, 1930, …, 2015].

The grouping process involved two stages. First, tables within the same database and schema were identified by removing sequences of four or more digits from names (e.g., PLACES_2018 → PLACES_), revealing common base names. Additional pattern checks ensured slight name variations still matched. Tables that shared the same base name were treated as candidate matches.

Second, each candidate group was verified for structural similarity by comparing column names, types, and description similarity (using SequenceMatcher with a threshold of 0.9). Tables were excluded if their structures or descriptions differed.

Below, I provide two examples of a group of similar tables that were successfully matched.


```
{
  "Key": "FEC/CENSUS_BUREAU_ACS",
  "Template": "CENSUSTRACT_201{variable}_SYR.json",
  "Variables": ["variable0", "0", "1", "2", "3", "4", "5", "6", "7"],
  "Grouped": true,
  "FilePaths": [
    "FEC/CENSUS_BUREAU_ACS/CENSUSTRACT_2010_SYR.json",
    "FEC/CENSUS_BUREAU_ACS/CENSUSTRACT_2011_SYR.json",
    "FEC/CENSUS_BUREAU_ACS/CENSUSTRACT_2012_SYR.json",
    "FEC/CENSUS_BUREAU_ACS/CENSUSTRACT_2013_SYR.json",
    "FEC/CENSUS_BUREAU_ACS/CENSUSTRACT_2014_SYR.json",
    "FEC/CENSUS_BUREAU_ACS/CENSUSTRACT_2015_SYR.json",
    "FEC/CENSUS_BUREAU_ACS/CENSUSTRACT_2016_SYR.json",
    "FEC/CENSUS_BUREAU_ACS/CENSUSTRACT_2017_SYR.json"
  ],
  "Combinations": [ ["0"], ["1"], ["2"], ["3"], ["4"], ["5"], ["6"], ["7"] ],
  "VariableOrder": ["variable0"]
}
```

```
{
  "Key": "NEW_YORK_GEO/NEW_YORK",
  "Template": "TLC_YELLOW_TRIPS_20{variable}.json",
  "Variables": ["variable0", "09", "10", "11", "12", "13", "14", "15", "16"],
  "Grouped": true,
  "FilePaths": [
    "NEW_YORK_GEO/NEW_YORK/TLC_YELLOW_TRIPS_2009.json",
    "NEW_YORK_GEO/NEW_YORK/TLC_YELLOW_TRIPS_2010.json",
    "NEW_YORK_GEO/NEW_YORK/TLC_YELLOW_TRIPS_2011.json",
    "NEW_YORK_GEO/NEW_YORK/TLC_YELLOW_TRIPS_2012.json",
    "NEW_YORK_GEO/NEW_YORK/TLC_YELLOW_TRIPS_2013.json",
    "NEW_YORK_GEO/NEW_YORK/TLC_YELLOW_TRIPS_2014.json",
    "NEW_YORK_GEO/NEW_YORK/TLC_YELLOW_TRIPS_2015.json",
    "NEW_YORK_GEO/NEW_YORK/TLC_YELLOW_TRIPS_2016.json"
  ],
  "Combinations": [ ["09"],["10"], ["11"], ["12"], ["13"], ["14"], ["15"], ["16"] ],
  "VariableOrder": ["variable0"]
}
```


Figure 7 JSON file examples of Data Preprocessing Phase Output

5.1.2 Data Augmentation

a Input

The input for this phase is the JSON file produced in the previous step. It includes both grouped templates and ungrouped (standalone) tables, along with all available metadata associated with their respective columns.

b Output

The output is an augmented JSON file that retains all original information while adding LLM-generated metadata. This includes comprehensive table and column descriptions, covering previously undocumented columns, together with a list of inferred keywords for each column and table. This enriched metadata serves as the foundation for the next stage of the pipeline, as outlined in Section 5.2.

c Process

This phase addresses missing table-level descriptions and inconsistent column documentation. To fill these gaps, I employed a large language model (gpt-4o-mini-2024-07-18) to generate both table and column descriptions, along with relevant keywords (see Figure 8). API calls were batched to reduce costs (by about 50%), with stream processing as a fallback in case of failures.

For each table, a table-level description and keywords were generated first (see Sections 10.4.1 & 10.4.2), followed by structured prompts for column descriptions and keywords. Prompts included the table description and as many column names as possible within token limits and were adapted depending on whether a table was grouped or standalone.

To clarify the schema's completeness, each prompt included a note indicating whether all columns were listed, only a subset was provided, only column names were shown without descriptions, or no other columns were included due to token limits.

Prompt examples are in the Appendix (see Sections 10.5.1 & 10.5.2). This design enabled consistent, contextually relevant outputs even with partial data. Additionally, the model returned keywords summarizing each column and table, supporting the retrieval stage (see Section 5.3.1). Figure 8 illustrates sample column outputs, and the final JSON structure is shown in Figure 5 (see Section 5.1).

```
{
 "ehail_fee": {
        "Description": "The ehail_fee represents an additional charge applied to electronic hails for rides booked through a mobile application or dispatch system.",
        "Keywords": [  "e-hail charge", "ride-hailing fee",  "app-based fare", "digital taxi fee", "mobile app charge" … ]
},
 "time_between_service": {
        "Description": "This column represents the time duration in seconds between the end of one taxi trip and the start of the next service trip for a given vehicle,
                providing insights into operational efficiency and wait times.",
        "Keywords": [ "service interval", "time gap", "duration between rides", "time between trips" … ]
  }
}
```

Figure 8 LLM Descriptions and Keywords Output

5.2 Vector Database

The following step is crucial for my pipeline, as it serves two essential functions: it constitutes the knowledge base on which my model relies, and it retrieves a selection of information through which the model should answer the given question. This aligns with the nature of the pipeline, which utilises the concept of Retrieval Augmented Generation (RAG), a hybrid approach that combines the "closed-book" parametric-memory of LLMs with "open-book" non-parametric memory retrieval techniques (Piktus et al., 2021, pp. 1-5).

The Vector Database Creation phase can be fundamentally divided into two distinct components: the generation of embeddings, and the construction of the vector database itself. Both steps are essential to ensure strong performance in the retrieval phase (see Section 5.3.1).

5.2.1 Vector Embeddings

After generating the JSON file with table and column descriptions, the next step is to create embeddings for efficient retrieval without exceeding the LLM's context window (see Section 5.3.1). To achieve this, I use two types of embeddings: dense and sparse.

Dense embeddings were produced using OpenAI's text-embedding-3-small model (1536-dimensional vectors) for both table- and column-level descriptions. Sparse embeddings were created using BM25 on column descriptions.

The rationale for embedding both tables and columns in the dense representation is to capture the wider context in which each column is set, namely its database, schema, and table environment, rather than relying solely on its description. My analysis showed substantial variation in column-level content and phrasing, making it meaningful to combine information with limited loss of information. Therefore, I constructed hybrid embeddings for each column using a weighted combination: 70% table embedding and 30% column embedding. This balances structural context and specific column semantics.

The final step in this phase is building the vector database. I chose Qdrant for its generous free tier, excellent documentation on query structuring and reranking ("Reranking in hybrid search," n.d.), and strong hybrid search performance (Łukawski, 2024).

In Qdrant, the top-level unit is a cluster, which can contain multiple collections. Each collection is a "named set of points" ("Collections," n.d.) that can be retrieved by vector similarity. Each point includes a dense embedding, a sparse embedding, and a payload (see Sections 5.1 & 5.2.1). The payload holds metadata used for both retrieval and answer generation, including database, schema, table template and variables, table keywords and description, as well as column name, type, description, sample values, and keywords.

While the embeddings power hybrid search, the payload supports conditional filtering to speed up retrieval (see Section 5.3.1) and provides the content needed to generate final answers (See Section 5.3.2 & 5.3.3).

5.3 Question-Answering Phase

The question-answering phase combines all components developed in the earlier stages of the pipeline. It is based on the Retrieval-Augmented Generation (RAG) framework, which integrates external, non-parametric sources of input to improve the capabilities of LLMs (Gao et al., 2024). In my implementation, I extend this framework by letting two large language models, each with a dedicated role, converse with one another to solve the task collaboratively. Even though the Spider 2.0 dataset is made to mimic the complexity and structure of real-world enterprise databases, it is static and does not evolve over time. In contrast, real-world databases are constantly changing. Therefore, it is important to consider mechanisms that enable real-time updates, extension, and examination, such as those made possible by vector databases (see Section 5.2). These capabilities make the system more adaptable to evolving information needs and enable more accurate, up-to-date responses (Piktus et al., 2020, p. 1).

In the following sections, I will explain in detail all the steps involved in the pipeline as shown in Figure 9. First, I will cover the retrieval phase of the pipeline (See Section 5.3.1), including paraphrasing, hybrid search against the vector database, and reranking. Next, I will focus on prompt construction (See Section 5.3.2), which encompasses a column-consistency check, sample retrieval, and the insertion of external knowledge. Finally, I will describe the conversational loop between two LLMs that produces the final query for the NL question (See Section 5.3.3).

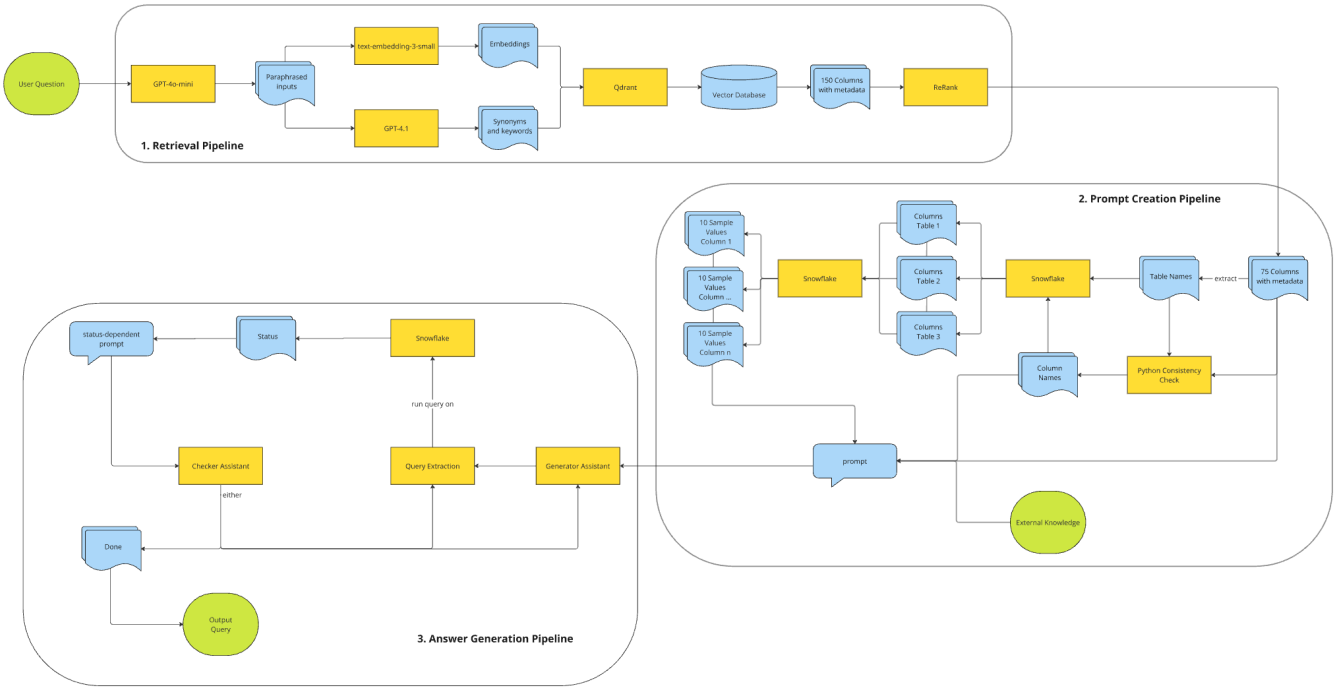

Figure 9 Question-Answering Phase Pipeline Overview

5.3.1 Retrieval Pipeline

The following section explain how the framework identifies the columns and tables needed to answer a question. This phase starts with a natural-language query and ends with a set of 75 columns drawn from multiple tables that are most closely associated with the query.

First, the user's question is paraphrased by GPT-4o-mini (see prompt in Appendix Section 10.6) to generate five variations: three at low temperature, one at medium (0.5), and one at high (0.7). These paraphrases help surface additional relevant columns while preserving the original intent.

Next, each paraphrase is processed in parallel. Dense embeddings are generated using text-embedding-3-small, while GPT-4.1 extracts structured signals such as entities (e.g., location, date, score), synonyms, aggregation phrases, filters, and domain-specific keywords that help clarify the user's intent (see prompt in Appendix, Section 10.7).

These signals are used to build "should" filters (boosting columns containing matching keywords) and "must" filters (e.g., database constraints) for hybrid retrieval in Qdrant (Qdrant, n.d.). From each search, the top 150 columns are initially retrieved to ensure high recall. Although this set is too large for the LLM context, retaining it ensures nothing is missed.

An LLM-based reranker (Cohere, n.d.) then reorders these columns by contextual relevance, and the top 75 are kept per paraphrase. After processing all five paraphrases, the resulting top-75 lists are merged, deduplicated, and finalized as the single set of most relevant columns.

In short, the system transforms the original question into multiple paraphrases, generates embeddings and structured signals, performs hybrid retrieval with filtering, reranks results, and consolidates them into a refined set of columns optimized for accuracy and coverage.

5.3.2 Prompt Creation Pipeline

This section describes how I verify the retrieved columns and create a precise prompt for the LLM. Starting from the 75 columns identified in the previous phase, I perform consistency checks on column names, tables, and sample values before integrating them into the prompt.

The first step is the Python Consistency Check. I extract all tables containing the retrieved columns, then query Snowflake to obtain authoritative lists of columns for each table. I ensure each column from retrieval exactly matches these lists (including case), preventing errors from invalid identifiers in Snowflake, which is case-sensitive.

Next, I retrieve 10 sample values for each column and table using a “distinct + limit query”. Providing these sample values helps the LLM understand possible conditions more precisely. For example, in question sf_local015 about helmet usage in motorcycle accidents:

Help me respectively calculate the percentage of motorcycle accident fatalities involving riders who were wearing helmets and those who weren't?

Providing all relevant distinct values (e.g., "driver, motorcycle helmet used") clarifies which conditions to apply and helps prevent invalid or illogical query construction (see Example in Appendix Section 10.8).

This step mirrors real-world scenarios, where table structures and values may change over time. Once validated, the information is used to build the LLM prompt (see Appendix, Section 10.1).

The prompt design explicitly specifies column names, expected output format, and includes an example output to guide query generation, as this is implicitly required by the pipeline’s evaluation setup (see Section 6.2). Additionally, learned guidelines, such as wrapping column names in double quotes, are provided to reduce errors. When available, relevant external knowledge is appended at the end of the prompt.

Finally, this prompt evolves during execution, adapting to different conditions in later stages (see Section 5.3.3).

5.3.3 Answer Generation Pipeline

This section outlines how the pipeline answers a user question by integrating three components: model-to-model dialogue, Snowflake database interactions, and an iterative loop of prompt design and output evaluation.

The procedure begins by instantiating two LLM Assistants using OpeanAI’s o3-mini-2025-01-03 model. One is the Generator Assistant, whose main role is to generate the SQL queries that will constitute the output of the pipeline; the other is the Checker Assistant, responsible for verifying that the generated SQL query accurately reflects the original question and that the returned results are correct and relevant.

Each Assistant is configured with a system prompt (defining its role and task, see Appendix Sections 10.2 & 10.3) and a user prompt (containing the natural language question). Unlike standard API calls that lack continuity, Assistants maintain conversational memory, enabling context retention across turns, similar to a persistent ChatGPT session.

I chose the o3-mini-2025-01-31 model for its strong reasoning capabilities and cost efficiency during testing. Later, as OpenAI reduced pricing for o3, I added a script to bypass the Assistants module and directly call the full o3 model (see Section 8).

Once the Generator receives the prompt section (from Section 5.3.2), it produces an SQL query. This query is then executed on Snowflake. If the output is malformed or lacks the expected START QUERY and END QUERY delimiters, the system retries up to three times or sends it back to the Assistant with a specific correction prompt (e.g. “Missing SQL Delimiters”).

Finally, after extracting a valid query, I attempt to run it in Snowflake and classify the result into one of four execution states.

a Success or Empty

If the query executes successfully and returns rows, I construct a prompt for the Checker Assistant that includes the original question, the SQL query, a sample of the result set, and the percentage of the full output shown, in order to keep the LLM’s context bounded. The Checker then assesses whether the query and its output genuinely answer the question. If it replies “Done,” the query is marked as correct. If it determines that adjustments are needed, either because the result set is empty or the output does not align with the question, it returns revision instructions. These are then passed to the Generator Assistant, which produces an updated query.

b Error

If a query fails to execute in Snowflake, I distinguish between two error classes: invalid-identifier errors and general errors. For invalid-identifier errors, I build a prompt for the Checker Assistant that includes the question, the faulty query, the Snowflake error message, and the database structure retrieved during prompt creation (see Section 5.3.2), meaning the relevant table and column names. For general errors, I use a nearly identical prompt but omit the schema details.

At this point, the Checker has two options: it can return a revised SQL query, which I immediately run and evaluate as usual, or it can provide step-by-step instructions for the Generator Assistant. If the Checker proposes a query and it fails again, I send it a follow-up prompt that explicitly forbids writing a new query and instead asks only for corrective guidance. These instructions are always based on the most recent query attempt, the Checker’s, so the next prompt to the Generator includes that query along with the Checker’s comments. If the Checker’s query succeeds, the results are passed back for validation; depending on the Checker’s verdict, the pipeline either terminates (if the answer is accepted) or loops again with new instructions for the Generator.

This dual role for the Checker, sometimes fixing the query directly, sometimes coaching the Generator, reduces both latency and cost, since it avoids unnecessary round-trips and duplicate token usage when the Checker already knows how to resolve the issue.

c Timeout

The final scenario occurs when a query runs on Snowflake longer than the user-specified time limit. In such cases, I terminate the execution and treat the query as provisionally correct. Because no results are returned, the Checker Assistant receives a prompt asking it to optimise the query so it completes within the allowed time. Since this execution produced no errors, the original query remains a candidate answer and, if no better alternative is found, it will be retained as the final output. It is important to note that long runtimes can result from queue delays rather than inefficient queries, so I measure only active execution time and exclude any time spent in the queue.

Every time a query is either successfully executed or times out, it is saved as a draft for potential inclusion in the final output. If no superior query is produced, this draft becomes the final answer. To manage both time and cost, the Generator-Checker exchange is capped at three iterations.

6 Results

6.1 Dataset

The pipeline is evaluated on the Spider 2.0 benchmark (Lei et al., 2024), which comprises three sub-tasks: Spider 2.0-Snow, Spider 2.0-Lite, and Spider 2.0-DBT. This work concentrates on Spider 2.0-Snow, a setting restricted to the Snowflake SQL dialect. The task presents 547 questions spanning more than 150 databases, each averaging roughly 800 columns (Deng et al., 2025). Designed to push beyond the difficulty of existing text-to-SQL challenges, Spider 2.0-Snow features much longer contexts and SQL queries that regularly exceed 100 lines (≈150 tokens on average), demanding more complex reasoning (Lei et al., 2024).

To reduce costs, I evaluated only a stratified subset of 405 out of 547 questions (≈ 74 %). I first selected the same proportion of queries from every database to keep the domain mix intact. Within each database slice, I balanced for difficulty along two quick axes: an LLM scored each question from 1 to 5 based on its wording length and the amount of reasoning it demands, and a script counted how many tables surfaced in the hybrid vector-database search to estimate schema load. Sampling across these two scores produced a lean yet still representative test set.

6.2 Evaluation

Pipeline performance is assessed with Execution Accuracy via the official Spider 2.0 evaluation script. A query is deemed correct only when column names and each returned value (tolerance ± 0.01) exactly match the ground truth. Compared with stricter benchmarks such as BIRD, this setting is more

forgiving: extra output columns are allowed, and column order is ignored unless the question explicitly requires it (Lei et al., 2024; Deng et al., 2025).

6.3 Large Language Models Used

The pipeline employs two LLMs selected for their availability, cost, and performance. For sentence paraphrasing, I rely on gpt-4o-mini because its results match those of costlier models on this narrow task; I vary its temperature to generate diverse question paraphrases. The second model is o3-mini, chosen for its reasoning strength and seamless integration with the OpenAI Assistants framework; at test time its usage cost was also very low. Unlike gpt-4o-mini, o3-mini does not accept an explicit temperature setting.

6.4 Results

The Spider 2.0-Snow benchmark has attracted only limited attention so far. To date, the best reported execution-accuracy scores are 38.21, 36.20, and 31.26, obtained with Relational Knowledge Graphs, ReFoRCE o3, and ReForRCE o1-preview, respectively (Lei et al., 2024). Earlier state-of-the-art text-to-SQL frameworks, DAIL-SQL, CHESS, and DIN-SQL, reached just 2.20, 1.28, and 0.00 execution accuracy, respectively (Lei et al., 2024).

My evaluation on the stratified subset (see Section 6.1) produced an execution accuracy of 33.24%. If I take a deliberately conservative stance and treat every one of the unanswered questions in the remaining 26 % of the benchmark as incorrect, the effective accuracy over the full dataset is 23.03%. The accuracy on the stratified subset trails the very top scores, and it surpasses many previous frameworks. Crucially, the leading results rely on the highest-capacity models (o3 and o1-preview) where a gap of roughly five percentage points shows that simply upgrading the base model already yields a sizable boost. In contrast, my pipeline delivers competitive performance with the leaner, and far cheaper, o3-mini model by combining a different framework with meticulous data preprocessing and retrieval.

Cost efficiency is another key advantage. ReFoRCE, itself an improvement over Spider-Agent, averages 3.52 v LLM calls and 3.89 v database calls per question, each LLM call consuming around 15K tokens (with v the number of concurrent voting candidates) (Deng et al., 2025). Spider-Agent required 11 v LLM calls at roughly 8K tokens each (Deng et al., 2025). By comparison, my method caps usage at 6 LLM calls and 3 database calls per question, with a total of about 70 K input tokens, an overall reduction in both runtime and cost relative to previously proposed frameworks.

Furthermore, the framework looks beyond the dataset's original scope: it adds a mechanism to check whether database details, especially column and table names, have drifted over time. This safeguard is crucial for the real-world settings that Spider 2.0-Snow aims to emulate.

7 Conclusion

This thesis set out to determine whether a lean retrieval-augmented generation (RAG) pipeline, powered mainly by the low-cost OpenAI o3-mini model, could tackle the challenging Spider 2.0-Snow benchmark. Based on a stratified review of 405 questions, this lightweight framework shows it can indeed hold its own. Despite consuming fewer tokens and calls, and relying on a model whose usage costs are dramatically lower than those behind the current record, the system still achieved an execution accuracy of 33.24%.

The study emphasises the significant influence that meticulous retrieval and schema curation has on performance, which goes beyond simple scores. By condensing near-duplicate tables into parameterised templates, enriching undocumented columns with LLM-generated descriptions, and storing both dense and sparse vectors in a hybrid Qdrant index, the language model spends its context window on schema fragments that actually matter. Just as pivotal is the dual-agent architecture: a Generator drafts SQL, while a Checker validates it, diagnoses failures, and, when needed, proposes fixes.

From a research standpoint, the work reframes Spider 2.0: the limiting factor is not only model size but the quality and selectivity of the context I feed it. It offers a reproducible, end-to-end evaluation template for live-database interaction and provides empirical support for peer-review style agent.

Moreover, it would be especially interesting to adopt a lean method in contexts where budget constraints or scalability concerns are critical, for example, within companies that want to empower both coders and non-coders to query data faster and more efficiently. This approach could serve as an accessible starting point for creating queries at a fraction of the cost of current state-of-the-art solutions.

Limitations remain, and Section 8 sketches concrete directions for improvement. Nonetheless, the central claim stands: enterprise-grade text-to-SQL is attainable with careful retrieval, thorough verification, and a cost-effective model selection.

8 Limitations and Future Research

While the project delivers a RAG-based LLM pipeline that competes with state-of-the-art systems on Spider 2.0-Snow and even goes beyond the dataset's original scope, several limitations remain.

A first priority is to test the framework on the full Spider 2.0-Snow set; my present results rest on a stratified subset, so whole-set validation would firm up the claims. In the same vein, my assessment still hinges on a single metric, Execution Accuracy. Augmenting the scoring information with partial-match or semantic-equivalence measures would give a richer view of model behaviour, especially for answers that are semantically correct but differ slightly in formatting or order.

Equally important is introducing a logic similar to the one in ReFoRCE (Deng et al., 2025), that is, creating several simultaneous threads built on the paraphrased questions and then applying majority voting to settle on the final SQL. Such parallelism could lift both robustness and accuracy.

Additionally, the model still runs on o3-mini, and it would be extremely interesting to swap in a more advanced model, plain o3 for example, while bypassing Assistants and their conversation state to see how much raw model capacity alone can add.

In the same spirit, one could implement a feedback store: after each query is verified as correct, the query would be cached and offered to the LLM when a similar question appears, allowing the system to evolve without fine-tuning. This idea extends on an experiment done in ReFoRCE (Deng et al., 2025), where simply providing the gold schema yields an absolute 8.26-point gain in Execution Accuracy.

Moreover, user interaction is strictly one-shot: the system never seeks clarification from the user. Adding an optional clarification loop, allowing the LLM to pose follow-up questions when a prompt is ambiguous, would resolve many edge cases. For instance, in question sf_local195 (Lei et al., 2024) the user asks, “Please find out how widespread the appeal of our top five actors is. What percentage of our customers have rented films featuring these actors?” yet “top five actors” could be defined by either the number of movies they are featured in or the total revenue they generated.

Finally, the pipeline depends on LLM-generated column and table descriptions. If those synthetic descriptions are off the mark, the model can inherit faulty assumptions. Improving or validating that metadata, perhaps via human spot-checks or automated consistency tests, will be essential for scaling the framework to production-grade, ever-changing databases.

9 Bibliography

Ahmad, K. (2008). *A method for mapping XML DTD to relational schemas in the presence of functional dependencies* (Doctoral dissertation, Universiti Putra Malaysia). Universiti Putra Malaysia Institutional. http://psasir.upm.edu.my/id/eprint/5247/1/FSKTM_2008_15a.pdf

Abdel-Aziz, A. A., & Oakasha, H. (2005). Mapping XML DTDs to relational schemas. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications* (pp. 47–). IEEE. https://doi.org/10.1109/AICCSA.2005.1387044

Alwan, A. A., Alzeber, M., Nordin, A., & Abualkishik, A. Z. (2017). A survey of schema matching research using database schemas and instances. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 8(10). https://pdfs.semanticscholar.org/0402/49f5bf2a40d5fc219e31ab286e96c36f441f.pdf

Androutsopoulos, I., Ritchie, G. D., & Thanisch, P. (1995). Natural language interfaces to databases - an introduction. *Natural Language Engineering*, 1(1), 29–81. https://doi.org/10.1017/S135132490000005X

Asif-Ur-Rahman, M., Hossain, B. A., Bewong, M., Islam, M. Z., Zhao, Y., Groves, J., & Judith, R. (2023). A semi-automated hybrid schema matching framework for vegetation data integration. *Expert Systems with Applications*, 229, Article 120405. https://doi.org/10.1016/j.eswa.2023.120405

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., & Sutton, C. (2021). Program synthesis with large language models. *arXiv.* https://doi.org/10.48550/arXiv.2108.07732

Batini, C., Lenzerini, M., & Navathe, S. B. (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4), 323–364. https://doi.org/10.1145/27633.27634

Benkley, S., Fandozzi, J., Housman, E., & Woodhouse, J. (1995). *Data element tool-based analysis (DELTA)* (MITRE Technical Report MTR'95B147). The MITRE Corporation.

Berant, J., & Liang, P. (2014, June). Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1415–1425). Baltimore, MD: Association for Computational Linguistics. https://aclanthology.org/P14-1133

Bergamaschi, S., Castano, S., & Vincini, M. (1999). Semantic integration of semistructured and structured data sources. *ACM SIGMOD Record*, 28(1), 54–59. https://doi.org/10.1145/309844.309897

Berlin, J., & Motro, A. (2002). Database schema matching using machine learning with feature selection. In A. B. Pidduck, M. T. Ozsu, J. Mylopoulos, & C. C. Woo (Eds.), *Advanced information systems engineering: 14th international conference, CAiSE 2002, Toronto, Canada, May 27–31, 2002, proceedings* (Lecture Notes in Computer Science, Vol. 2348, pp. 452–466). Springer, Berlin Heidelberg. https://doi.org/10.1007/3-540-47961-9_32

Bilke, A., & Naumann, F. (2005, April). Schema matching using duplicates. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)* (pp. 69–80). IEEE. https://doi.org/10.1109/ICDE.2005.126

Boerrigter, T. (2021). *Intelligence amplification framework for schema matching in enterprise iPaaS* (Master's thesis, University of Twente). University of Twente Repository. https://essay.utwente.nl/88064/

Bogin, B., Gardner, M., & Berant, J. (2019). Representing schema structure with graph neural networks for text-to-SQL parsing. *arXiv*. https://doi.org/10.48550/arXiv.1905.06241

Borchmann, Ł., & Wydmuch, M. (2025). Query and conquer: Execution-guided SQL generation. *arXiv*. https://doi.org/10.48550/arXiv.2503.24364

Bounif, H. (2007). Predictive approach for database schema evolution. In J. Erickson & K.-P. Yoon (Eds.), *Intelligent databases: Technologies and applications* (pp. 307–330). IGI Global. https://doi.org/10.4018/978-1-59904-120-9.ch011

Cao, Z., Zheng, Y., Fan, Z., Zhang, X., Chen, W., & Bai, X. (2024). RSL-SQL: Robust schema linking in text-to-SQL generation. *arXiv*. https://doi.org/10.48550/arXiv.2411.00073

Cappuzzo, R., Papotti, P., & Thirumuruganathan, S. (2020, June). Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (pp. 1335–1349). ACM. https://doi.org/10.1145/3318464.3389742

Cen, J., Liu, J., Li, Z., & Wang, J. (2025). SQLFixAgent: Towards semantic-accurate text-to-SQL parsing via consistency-enhanced multi-agent collaboration. *Proceedings of the AAAI Conference on Artificial Intelligence, 39*(1), 49–57. https://doi.org/10.1609/aaai.v39i1.31979

Chasseur, C., Li, Y., & Patel, J. M. (2013). Enabling JSON document stores in relational systems. *Proceedings of the WebDB 2013 Conference*. https://pages.cs.wisc.edu/~chasseur/pubs/argo-long.pdf

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Harri Edwards, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., … Zaremba, W. (2021). *Evaluating large language models trained on code. arXiv*. https://doi.org/10.48550/arXiv.2107.03374

Chen, W., Ma, X., Wang, X., & Cohen, W. W. (2022). Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv*. https://doi.org/10.48550/arXiv.2211.12588

Chen, Z., Narasayya, V. R., & Chaudhuri, S. (2014). Fast foreign-key detection in Microsoft SQL Server PowerPivot for Excel. *Proceedings of the VLDB Endowment, 7*(13), 1417–1428. https://doi.org/10.14778/2733004.2733014

Chronas, K. (2023). *Generating labeled datasets for schema matching* (Master's thesis, Delft University of Technology). TU Delft Repository. https://repository.tudelft.nl/file/File_f2f8f42d-ef77-4e26-90d1-75bf4f432a99

Chua, C. E. H., Chiang, R. H., & Lim, E. P. (2003). Instance-based attribute identification in database integration. *The VLDB Journal*, *12*, 228-243.

Chua, C. E. H., Chiang, R. H. L., & Lim, E. P. (2003). Instance-based attribute identification in database integration. *The VLDB Journal, 12*, 228–243. https://doi.org/10.1007/s00778-003-0088-y

Cilibrasi, R. L., & Vitanyi, P. M. B. (2007). The Google similarity distance. *IEEE Transactions on Knowledge and Data Engineering, 19*(3), 370–383. https://doi.org/10.1109/TKDE.2007.48

Cohere. (n.d.). *Rerank | Boost Enterprise Search and Retrieval*. Retrieved July 5, 2025, from https://cohere.com/rerank

Cruz, I. F., Antonelli, F. P., & Stroe, C. (2009). AgreementMaker: Efficient matching for large real-world schemas and ontologies. In *Proceedings of the VLDB Endowment, 2*(2), 1586–1589. https://doi.org/10.14778/1687553.1687598

Deng, M., Ramachandran, A., Xu, C., Hu, L., Yao, Z., Datta, A., & Zhang, H. (2025). ReFoRCE: A text-to-SQL agent with self-refinement, format restriction, and column exploration. *arXix*. https://doi.org/10.48550/arXiv.2502.00675

Dhamankar, R., Lee, Y., Doan, A., Halevy, A., & Domingos, P. (2004, June). iMAP: Discovering complex semantic matches between database schemas. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (pp. 383–394). Association for Computing Machinery. https://doi.org/10.1145/1007568.1007612

Doan, A., & Halevy, A. Y. (2005). Semantic integration research in the database community: A brief survey. *AI Magazine, 26*(1), 83. https://doi.org/10.1609/aimag.v26i1.1801

Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., & Halevy, A. (2003). Learning to match ontologies on the Semantic Web. *The VLDB Journal, 12*(4), 303–319. https://doi.org/10.1007/s00778-003-0104-2

Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Lin, J., & Lou, D. (2023). C3: Zero-shot text-to-SQL with ChatGPT. *arXiv.* https://doi.org/10.48550/arXiv.2307.07306

Dong, Z., Sun, S., Liu, H., Lou, J. G., & Zhang, D. (2019, November). Data-anonymous encoding for text-to-SQL generation. In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 5405–5414). Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1543

Engle, J., Feng, Y., & Goldstone, R. *Inducing Relatedness Graphs for Data Integration*. Indiana University.

Engmann, D., & Massmann, S. (2007, March). Instance matching with COMA++. In *BTW workshops* (Vol. 7, pp. 28–37).

Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal, 10*(4), 334–350. https://doi.org/10.1007/s007780100057

Fagin, R. (1979). Normal forms and relational database operators. *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, 153–160. https://doi.org/10.1145/582095.582120

Fagin, R., Haas, L. M., Hernández, M., Miller, R. J., Popa, L., & Velegrakis, Y. (2009). Clio: Schema mapping creation and data exchange, 198-236.

Fang, X., Xu, W., Tan, F. A., Zhang, J., Hu, Z., Qi, Y., Nickleach, S., Socolinsky, D., Sengamedu, S., & Faloutsos, C. (2024, March 1). Large Language Models(LLMs) on Tabular Data: Prediction, Generation, and Understanding - A Survey. https://arxiv.org/pdf/2402.17944

Fellbaum, C. (2010). WordNet. In R. Poli, M. Healy, & A. Kameas (Eds.), *Theory and applications of ontology: Computer applications* (pp. 231–243). Springer.

Feng, J., Hong, X., & Qu, Y. (2009, August). An instance-based schema matching method with attributes ranking and classification. In *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery* (Vol. 5, pp. 522-526). IEEE. https://ieeexplore.ieee.org/abstract/document/5360566

Fernandez, R. C., Abedjan, Z., Koko, F., Yuan, G., Madden, S., & Stonebraker, M. (2018, April). Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)* (pp. 1001-1012). IEEE. https://ieeexplore.ieee.org/abstract/document/8509315

Fernandez, R. C., Mansour, E., Qahtan, A. A., Elmagarmid, A., Ilyas, I., Madden, S., ... & Tang, N. (2018, April). Seeping semantics: Linking datasets using word embeddings for data discovery.

In *2018 IEEE 34th International Conference on Data Engineering (ICDE)* (pp. 989-1000). IEEE. https://ieeexplore.ieee.org/abstract/document/8509314

Floratou, A., Psallidas, F., Zhao, F., Deep, S., Hagleither, G., Tan, W., ... & Curino, C. (2024, January). NL2SQL is a solved problem... Not!. In CIDR

Floratou, A., Psallidas, F., Zhao, F., Deep, S., Hagleitner, G., Tan, W., Cahoon, J., Alotaibi, R., Henkel, J., Singla, A., van Grootel, A., Chow, B., Deng, K., Lin, K., Campos, M., Emani, V., Pandit, V., Shnayder, V., Wang, W., & Curino, C. (2024, January). NL2SQL is a solved problem... Not! In *Proceedings of the 14th Annual Conference on Innovative Data Systems Research (CIDR '24)*, Chaminade, USA. CIDR. https://www.vldb.org/cidrdb/papers/2024/p74-floratou.pdf

Fu, X., Shi, W., Yu, X., Zhao, Z., & Roth, D. (2020). Design challenges in low-resource cross-lingual entity linking. *arXiv*. https://arxiv.org/abs/2005.00692

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., & Zhou, J. (2023). Text-to-SQL empowered by large language models: A benchmark evaluation. *arXiv*. https://arxiv.org/abs/2308.15363

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Zhang, H., Zhang, Y., Liu, X., Liu, W., Wang, Y., & Wang, H. (2024, March 27). Retrieval-augmented generation for large language models: A survey. *arXiv*. https://arxiv.org/pdf/2312.10997.

Giunchiglia, F., Avesani, P., & Yatskevich, M. (2005). A large scale taxonomy mapping evaluation. In I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, & L. Aroyo (Eds.), *The Semantic Web – ISWC 2005* (pp. 67–81). Galway, Ireland. Springer. https://doi.org/10.1007/11574620_8

Giunchiglia, F., Shvaiko, P., & Yatskevich, M. (2004). S-Match: An algorithm and an implementation of semantic matching. In C. J. Bussler, J. Davies, D. Fensel, & R. Studer (Eds.), *The Semantic Web: Research and Applications. ESWS 2004* (Lecture Notes in Computer Science, Vol. 3053, pp. 61–75). Springer. https://doi.org/10.1007/978-3-540-25956-5_5

Glass, M., Eyceoz, M., Subramanian, D., Rossiello, G., Vu, L., & Gliozzo, A. (2025). Extractive schema linking for text-to-SQL. *arXiv*. https://arxiv.org/abs/2501.17174

Gligorov, R., Ten Kate, W., Aleksovski, Z., & van Harmelen, F. (2007). Using Google distance to weight approximate ontology matches. In *Proceedings of the 16th international conference on World Wide Web* (pp. 767–776). ACM. https://doi.org/10.1145/1242572.1242676

Gozudeli, Y., Karacan, H., Yildiz, O., Baker, M. R., Minnet, A., Kalender, M., Ozay, O., & Akcayol, M. A. (2015). A new method based on Tree simplification and schema matching for automatic web result extraction and matching. In *Proceedings of the International MultiConference of Engineers and Computer Scientists 2015, Vol. I* (IMECS 2015, March 18–20, 2015, Hong Kong) (pp. 369–373). IMECS. http://dit.unitn.it/~p2p/RelatedWork/Matching/IMECS2015_pp369-373.pdf

Grosz, B. J. (1983, February). TEAM: A transportable natural-language interface system. In *Proceedings of the First Conference on Applied Natural Language Processing* (pp. 39–45). Association for Computational Linguistics. https://doi.org/10.3115/974194.974201

Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J. G., Liu, T., & Zhang, D. (2019). Towards complex text-to-SQL in cross-domain database with intermediate representation. *arXiv*. https://arxiv.org/abs/1905.08205

Haas, L. M., Hernández, M. A., Ho, H., Popa, L., & Roth, M. (2005, June). Clio grows up: from research prototype to industrial tool. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data (pp. 805-810).

Haas, L. M., Hernández, M. A., Ho, H., Popa, L., & Roth, M. (2005, June). Clio grows up: from research prototype to industrial tool. In *Proceedings of the 2005 ACM SIGMOD International Conference*

on Management of Data (pp. 805–810). Association for Computing Machinery. https://doi.org/10.1145/1066157.1066252

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, & Preslav Nakov (13 August 2022). On the effect of dropping layers of pre-trained transformer models. https://arxiv.org/pdf/2004.03844

Hättasch, B., Truong-Ngoc, M., Schmidt, A., & Binnig, C. (2020, August 31). It's AI Match: A two-step approach for schema matching using embeddings. In *Proceedings of the 2nd International Workshop on Applied AI for Database Systems and Applications (AIDB '20)* (Tokyo, Japan). *arXiv*. https://arxiv.org/abs/2203.04366

He, Q., & Ling, T. W. (2004). Extending and inferring functional dependencies in schema transformation. In Proceedings of the thirteenth ACM international conference on Information and knowledge management (pp. 12–21). ACM. https://doi.org/10.1145/1031171.1031177

Herzig, J., & Berant, J. (2018). Decoupling structure and lexicon for zero-shot semantic parsing. *arXiv*. https://arxiv.org/abs/1804.07918

Hong, Z., Yuan, Z., Zhang, Q., Chen, H., Dong, J., Huang, F., & Huang, X. (2025). Next-generation database interfaces: A survey of llm-based text-to-sql. *arXiv*. https://arxiv.org/abs/2406.08426

Hristidis, V., Papakonstantinou, Y., & Gravano, L. (2003, January). Efficient IR-style keyword search over relational databases. In *Proceedings of the 29th VLDB Conference* (pp. 850–861). Morgan Kaufmann. https://doi.org/10.1016/B978-012722442-8/50080-X

Jiang, L., & Naumann, F. (2020). Holistic primary key and foreign key detection. *Journal of Intelligent Information Systems, 54*(3), 439–461. https://doi.org/10.1007/s10844-019-00562-z

Jin, Z., Tse, B., Chauhan, G., Sachan, M., & Mihalcea, R. (2023, January 18). How good is NLP? A sober look at NLP tasks through the lens of social impact. *arXiv*. https://arxiv.org/abs/2106.02359

Jose, M. A., & Cozman, F. G. (2023). A multilingual translator to SQL with database schema pruning to improve self-attention. *International Journal of Information Technology, 15*, 3015–3023. https://doi.org/10.1007/s41870-023-01342-3

K.K. Nambiar, B. Gopinath, T. Nagaraj, & S. Manjunath (1996). Boyce-Codd Normal Form Decomposition.

Kang, J., & Naughton, J. F. (2003, June). On schema matching with opaque column names and data values. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (pp. 205–216). ACM. https://doi.org/10.1145/872757.872784

Kang, J., & Naughton, J. F. (2008). Schema matching using interattribute dependencies. *IEEE Transactions on Knowledge and Data Engineering*, *20*(10), 1393–1407. https://doi.org/10.1109/TKDE.2008.100

Kolahi, S., & Libkin, L. (2008). An information-theoretic analysis of worst-case redundancy in database design. Proceedings of the ACM Symposium on Principles of Database Systems, 15 February 2008. https://dl.acm.org/doi/10.1145/1670243.1670248

Kolahi, S., & Libkin, L. (2010). An information-theoretic analysis of worst-case redundancy in database design. *ACM Transactions on Database Systems, 35*(1), Article 5. https://doi.org/10.1145/1670243.1670248

Koutras, C. (2024). *Tabular schema matching for modern settings* (Doctoral dissertation, Delft University of Technology). https://doi.org/10.4233/uuid:d6d859df-8e15-4f7b-9eef-10452c96bd36

Koutras, C., Fragkoulis, M., Katsifodimos, A., & Lofi, C. (2020, March). REMA: Graph Embeddings-based Relational Schema Matching. In Edbt/icdt workshops (p. 17).

Koutras, C., Fragkoulis, M., Katsifodimos, A., & Lofi, C. (2020, March). REMA: Graph embeddings-based relational schema matching. In *EDBT/ICDT workshops* (p. 17). CEUR Workshop Proceedings. http://star.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-2578/SEAData5.pdf

Kwiatkowski, T., Choi, E., Artzi, Y., & Zettlemoyer, L. (2013, October). Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 1545–1556). Association for Computational Linguistics.

Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. *Discourse Processes, 25*(2–3), 259–284. https://doi.org/10.1080/01638539809545028

Lee, D., Park, C., Kim, J., & Park, H. (2024). MCS-SQL: Leveraging multiple prompts and multiple-choice selection for text-to-SQL generation. *arXiv preprint*. https://arxiv.org/abs/2405.07467

Lei, F., Chen, J., Ye, Y., Cao, R., Shin, D., Su, H., Suo, Z., Gao, H., Hu, W., Yin, P., Zhong, V., Xiong, C., Sun, R., Liu, Q., Wang, S., & Yu, T. (2024). Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. arXiv. https://arxiv.org/abs/2411.07763

Lei, F., Chen, J., Ye, Y., Cao, R., Shin, D., Su, H., Suo, Z., Gao, H., Hu, W., Yin, P., Zhong, V., Xiong, C., Sun, R., Liu, Q., Wang, S., & Yu, T. (n.d.). Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. Retrieved June 22, 2025, from https://spider2-sql.github.io/

Lei, W., Wang, W., Ma, Z., Gan, T., Lu, W., Kan, M. Y., & Chua, T. S. (2020, November). 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)* (pp. 6943–6954). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.557

Leme, L. A. P. P., Casanova, M. A., Breitman, K. K., & Furtado, A. L. (2009). Instance-based OWL schema matching. In Enterprise Information Systems: 11th International Conference, ICEIS 2009, Milan, Italy, May 6-10, 2009. Proceedings 11 (pp. 14-26). Springer Berlin Heidelberg. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d222789382fd59cbf1b6dde14f54a2c98db421de

Leme, L. A. P. P., Casanova, M. A., Breitman, K. K., & Furtado, A. L. (2009). Instance-based OWL schema matching. In J. Filipe & J. Cordeiro (Eds.), *Enterprise Information Systems: 11th International Conference, ICEIS 2009, Milan, Italy, May 6-10, 2009. Proceedings* (Vol. 24, pp. 14–26). Springer Berlin Heidelberg. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d222789382fd59cbf1b6dde14f54a2c98db421de

Levy, M., Jacoby, A., & Goldberg, Y. (2024). Same task, more tokens: The impact of input length on the reasoning performance of large language models. *arXiv*. https://arxiv.org/pdf/2402.14848

Li, B., Luo, Y., Chai, C., Li, G., & Tang, N. (2024). The dawn of natural language to SQL: Are we fully ready? *Proceedings of the VLDB Endowment, 17*(11). https://doi.org/10.14778/3681954.3682003

Li, F., & Jagadish, H. V. (2014). Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment, 8*(1), 73–84.

Li, H., Zhang, J., Li, C., & Chen, H. (2023, June). Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 37, No. 11, pp. 13067-13075). . https://doi.org/10.1609/aaai.v37i11.26535

Li, H., Zhang, J., Liu, H., Fan, J., Zhang, X., Zhu, J., Wei, R., Pan, H., Li, C., & Chen, H. (2024). CodeS: Towards building open-source language models for text-to-SQL. *Proceedings of the ACM on Management of Data, 2*(3), Article 127, 1–28. https://doi.org/10.1145/3654930

Li, J., Hui, B., Cheng, R., Qin, B., Ma, C., Huo, N., Huang, F., Du, W., Si, L., & Li, Y. (2023, June). Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 37, No. 11, pp. 13076-13084). https://doi.org/10.1609/aaai.v37i11.26536

Li, J., Hui, B., Qu, G., Li, B., Yang, J., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K. C. C., Huang, F., Cheng, R., & Li, Y. (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* (Vol. 36, pp. 42330–42357). 37th Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks. *CoRR*, abs/2305.03111. https://proceedings.neurips.cc/paper_files/paper/2023/file/83fc8fab1710363050bbd1d4b8cc0021-Paper-Datasets_and_Benchmarks.pdf

Li, S., Zhou, K., Zhuang, Z., Wang, H., & Ma, J. (2023). Towards text-to-SQL over aggregate tables. *Data Intelligence*, *5*(2), 457–474. https://doi.org/10.1162/dint_a_00194

Li, W. S., & Clifton, C. (2000). SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Knowledge Engineering, 33*(1), 49–84. https://doi.org/10.1016/S0169-023X(99)00044-0

Li, Y., Li, J., Suhara, Y., Doan, A., & Tan, W. C. (2020). Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment, 14*(1), 1–13. https://doi.org/10.14778/3421424.3421431

Liang, Y. (2008, March). An instance-based approach for domain-independent schema matching. In *Proceedings of the 46th Annual ACM Southeast Conference* (pp. 268–271). ACM.

Lin, X. V., Socher, R., & Xiong, C. (2020). Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. *arXiv.* https://doi.org/10.48550/arXiv.2012.12627

Logeswaran, L., Chang, M. W., Lee, K., Toutanova, K., Devlin, J., & Lee, H. (2019). Zero-shot entity linking by reading entity descriptions. *arXiv preprint arXiv:1906.07348*. https://doi.org/10.48550/arXiv.1906.07348

Lopes, S., Petit, J.-M., & Toumani, F. (2002). Discovering interesting inclusion dependencies: Application to logical database tuning. *Information Systems, 27*(1), 1–19. https://doi.org/10.1016/S0306-4379(01)00027-8

Łukawski, K. (2024, July 25). *Hybrid search revamped — Building with Qdrant's Query API*. Qdrant. https://qdrant.tech/articles/hybrid-search/

Luoma, K., & Kumar, A. (2025, February). SNAILS: Schema naming assessments for improved LLM-based SQL inference. *Proceedings of the ACM on Management of Data, 3*(1, SIGMOD), Article 77, 26 pages. https://doi.org/10.1145/3709727

Ma, C., Chakrabarti, S., Khan, A., & Molnár, B. (2025). Knowledge Graph-based Retrieval-Augmented Generation for Schema Matching. *arXiv.* https://arxiv.org/abs/2501.08686

Maamari, K., Abubaker, F., Jaroslawicz, D., & Mhedhbi, A. (2024, August 18). *The death of schema linking? Text-to-SQL in the age of well-reasoned language models*. arXiv. https://arxiv.org/pdf/2408.07702

Maamari, K., Abubaker, F., Jaroslawicz, D., & Mhedhbi, A. (2024). The death of schema linking? Text-to-SQL in the age of well-reasoned language models. *arXiv.* https://doi.org/10.48550/arXiv.2408.07702

Madhavan, J., Bernstein, P. A., & Rahm, E. (2001, September). Generic schema matching with Cupid. In *Proceedings of the VLDB Conference* (Vol. 1, No. 2001, pp. 49–58).

Marchi, F. D., Lopes, S., & Petit, J. M. (2009). Unary and n-ary inclusion dependency discovery in relational databases. *Journal of Intelligent Information Systems, 32*(1), 53–73. https://doi.org/10.1007/s10844-007-0048-x

Mehdi, O., Ibrahim, H., & Affendey, L. (2017). An Approach for Instance Based Schema Matching with Google Similarity and Regular Expression. *International Arab Journal of Information Technology (IAJIT)*, 14(5).

Melnik, S., Garcia-Molina, H., & Rahm, E. (2002, February). Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering* (pp. 117–128). IEEE. https://doi.org/10.1109/ICDE.2002.994702

Memari, M., Link, S., & Dobbie, G. (2015). SQL data profiling of foreign keys. In P. Johannesson, M. Lee, S. Liddle, A. Opdahl, & Ó. Pastor López (Eds.), *Conceptual Modeling. ER 2015. Lecture Notes in Computer Science* (Vol. 9381, pp. 229–243). Springer, Cham. https://doi.org/10.1007/978-3-319-25264-3_17

Meurice, L., & Cleve, A. (2014). *DAHLIA: A visual analyzer of database schema evolution*. PReCISE Research Center, University of Namur, Belgium. Retrieved from https://loupmeurice.github.io/publications/CSMR-WCRE2014.pdf

Meurice, L., Bermúdez Ruiz, F. J., Weber, J. H., & Cleve, A. (2014). Establishing referential integrity in legacy information systems – reality bites! In *Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME)* (pp. 461–465). IEEE. https://doi.org/10.1109/ICSME.2014.74

Miller, R. J., Hernández, M. A., Haas, L. M., Yan, L., Ho, C. T. H., Fagin, R., & Popa, L. (2001). The Clio project: managing heterogeneity. *ACM SIGMOD Record, 30*(1), 78–83. https://doi.org/10.1145/373626.373713

Motl, J., & Kordík, P. (2017). Foreign Key Constraint Identification in Relational Databases. In J. Hlavácová (Ed.), *ITAT 2017 Proceedings* (pp. 106–111). CEUR Workshop Proceedings, Vol. 1885. https://ceur-ws.org/Vol-1885/106.pdf

Nguyen, X. B., Phan, X. H., & Piccardi, M. (2025). Fine-tuning text-to-SQL models with reinforcement-learning training objectives. *Natural Language Processing Journal, 10*, 100135. https://doi.org/10.1016/j.nlp.2025.100135

Palopoli, L., Terracina, G., & Ursino, D. (2003). DIKE: A system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. *Software: Practice and Experience, 33*(9), 847–884. https://doi.org/10.1002/spe.531

Papenbrock, T., & Naumann, F. (2017). Data-driven schema normalization. In *Proceedings of the 20th International Conference on Extending Database Technology (EDBT)*. Venice, Italy. https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/publications/PDFs/2017_papenbrock_datadriven.pdf

Paton, V., Gabor, A., Flores, R. O. R., Badia-i-Mompel, P., Tanevski, J., Garrido-Rodriguez, M., & Saez-Rodriguez, J. (2024). Assessing the impact of transcriptomics data analysis pipelines on downstream functional enrichment results. *Nucleic Acids Research, 52*(14), 8100–8111. https://doi.org/10.1093/nar/gkae552

Pavel Shvaiko, & Jerome Euzenat (2005). A Survey of Schema-Based Matching Approaches. https://exmo.inria.fr/files/publications/shvaiko2005a.pdf

Piktus, A., Lewis, P., Perez, E., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-

intensive NLP tasks. *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, Vancouver, Canada. https://arxiv.org/pdf/2005.11401

Popescu, A. M., Etzioni, O., & Kautz, H. (2003, January). Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces* (pp. 149–157). Miami, Florida, USA. ACM.

Pourreza, M., & Rafiei, D. (2023). DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. *Advances in Neural Information Processing Systems*, 36, 36339–36348. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*. University of Alberta. https://proceedings.neurips.cc/paper_files/paper/2023/hash/72223cc66f63ca1aa59edaec1b3670e6-Abstract-Conference.html

Pourreza, M., Talaei, S., Sun, R., Wan, X., Li, H., Mirhoseini, A., Saberi, A., & Arik, S. Ö. (2025). Reasoning-SQL: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-SQL. *arXiv.* https://doi.org/10.48550/arXiv.2503.23157

Qdrant. (n.d.). *Collections*. https://qdrant.tech/documentation/concepts/collections/

Qdrant. (n.d.). *Qdrant.* From https://qdrant.tech/

Qdrant. (n.d.). *Reranking in hybrid search.* https://qdrant.tech/documentation/advanced-tutorials/reranking-hybrid-search/

Qi, J., Tang, J., He, Z., Wan, X., Cheng, Y., Zhou, C., Wang, X., Zhang, Q., & Lin, Z. (2022). RASAT: Integrating relational structures into pretrained Seq2Seq model for text-to-SQL. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing* (pp. 3215–3229). Abu Dhabi, United Arab Emirates: Association for Computational Linguistics. https://doi.org/10.48550/arXiv.2205.06983

Qiang, Z., Taylor, K., & Wang, W. (2025). How does a text preprocessing pipeline affect ontology syntactic matching? Experiment, analysis & benchmark. *arXiv.* https://doi.org/10.48550/arXiv.2411.03962

Qu, G., Li, J., Li, B., Qin, B., Huo, N., Ma, C., & Cheng, R. (2024). Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation. arXiv. https://doi.org/10.48550/arXiv.2405.15307

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of machine learning research, 21(140), 1-67.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research, 21*(140), 1–67.

Rastogi, A., Hakkani-Tür, D., & Heck, L. (2017, December). Scalable multi-domain dialogue state tracking. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)* (pp. 561–568), Okinawa, Japan. IEEE. https://doi.org/10.1109/ASRU.2017.8268986

Ren, L., Xie, K., Chen, L., & Yu, K. (2018). Towards universal dialogue state tracking. arXiv preprint arXiv:1810.09587.

Ren, L., Xie, K., Chen, L., & Yu, K. (2018). *Towards universal dialogue state tracking*. *arXiv*. https://doi.org/10.48550/arXiv.1810.09587

Rijhwani, S., Xie, J., Neubig, G., & Carbonell, J. (2019). Zero-shot neural transfer for cross-lingual entity linking. *Proceedings of the AAAI Conference on Artificial Intelligence, 33*(01), 6924–6931. https://doi.org/10.1609/aaai.v33i01.33016924

Rong, S., Niu, X., Xiang, E. W., Wang, H., Yang, Q., & Yu, Y. (2012). A machine learning approach for instance matching based on similarity metrics. In P. Cudré-Mauroux et al. (Eds.), *Proceedings of the 11th International Semantic Web Conference (ISWC 2012)* (Lecture Notes in Computer Science, Vol. 7649, pp. 460–475). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35176-1_29

Rostin, A., Albrecht, O., Bauckmann, J., Naumann, F., & Leser, U. (2009). A machine learning approach to foreign key discovery. In *Proceedings of the 12th International Workshop on the Web and Databases (WebDB 2009)*. Retrieved from https://scispace.com/pdf/a-machine-learning-approach-to-foreign-key-discovery-5gfqlqnqs5.pdf

Sajjad, H., Dalvi, F., Durrani, N., & Nakov, P. (2023). On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language, 77*, Article 101429. https://doi.org/10.1016/j.csl.2022.101429

Scholak, T., Schucher, N., & Bahdanau, D. (2021, November 7–11). PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (pp. 9895–9901). *arXiv.* https://doi.org/10.48550/arXiv.2109.05093

Seedat, N., & van der Schaar, M. (2024). Matchmaker: Self-Improving Large Language Model Programs for Schema Matching. arXiv preprint arXiv:2410.24105.

Seedat, N., & van der Schaar, M. (2024). Matchmaker: Self-improving large language model programs for schema matching. In *GenAI for Health Workshop, NeurIPS 2024*, Vancouver. *arXiv*. https://doi.org/10.48550/arXiv.2410.24105

Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., & Naughton, J. (2008). Relational Databases for Querying XML Documents: Limitations and Opportunities. http://ocw.abu.edu.ng/courses/biological-engineering/20-453j-biomedical-information-technology-fall-2008/readings/inlining_vldb.pdf

Sheetrit, E., Brief, M., Mishaeli, M., & Elisha, O. (2024). *ReMatch: Retrieval enhanced schema matching with LLMs*. *arXiv.* https://doi.org/10.48550/arXiv.2403.01567

Shen, C., Wang, J., Rahman, S., & Kandogan, E. (2025). *MageSQL: Enhancing in-context learning for text-to-SQL applications with large language models*. *arXiv.* https://doi.org/10.48550/arXiv.2504.02055

Shi, T., Tatwawadi, K., Chakrabarti, K., Mao, Y., Polozov, O., & Chen, W. (2018). *IncSQL: Training incremental text-to-SQL parsers with non-deterministic oracles*. *arXiv*. https://doi.org/10.48550/arXiv.1809.05054

Shoval, P., & Even-Chaime, M. (1987). Database schema design: An experimental comparison between normalization and information analysis. *ACM SIGMIS Database: The DATABASE for Advances in Information Systems*, *18*(3), 30–39. https://doi.org/10.1145/27544.27548

Shraga, R., Gal, A., & Roitman, H. (2020). ADnEV: Cross-domain schema matching using deep similarity matrix adjustment and evaluation. *Proceedings of the VLDB Endowment*, *13*(9), 1401–1415. https://doi.org/10.14778/3397230.3397237

Shvaiko, P., Euzenat, J. (2005). A Survey of Schema-Based Matching Approaches. In: Spaccapietra, S. (eds) *Journal on Data Semantics IV*. Lecture Notes in Computer Science, vol 3730, pp. 146-171. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11603412_5

Stonebraker, M., Bruckner, D., Ilyas, I. F., Beskales, G., Cherniack, M., Zdonik, S. B., Pagan, A., & Xu, S. (2013, January). Data curation at scale: The Data Tamer system. In *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR 2013)* (Vol. 2013). Asilomar, California, USA.

Su, Y., & Yan, X. (2017). Cross-domain semantic parsing via paraphrasing. *arXiv.* https://doi.org/10.48550/arXiv.1704.05974

Sun, R., Arik, S. Ö., Muzio, A., Miculicich, L., Gundabathula, S., Yin, P., Dai, H., Nakhost, H., Sinha, R., Wang, Z., & Pfister, T. (2024). SQL-PaLM: Improved large language model adaptation for Text-to-SQL (extended). *arXiv.* https://doi.org/10.48550/arXiv.2306.00739

Sutanta, E., Wardoyo, R., Mustofa, K., & Winarko, E. (2016). A hybrid model schema matching using constraint-based and instance-based. *International Journal of Electrical & Computer Engineering (IJECE)*, *6*(3), 1048–1058. https://doi.org/10.11591/ijece.v6i3.9790

Talaei, S., Pourreza, M., Chang, Y. C., Mirhoseini, A., & Saberi, A. (2024). CHESS: Contextual harnessing for efficient SQL synthesis. *arXiv*. https://doi.org/10.48550/arXiv.2405.16755

Thorpe, D. G., Duberstein, A. J., & Kinsey, I. A. (2024). *Dubo-SQL: Diverse retrieval-augmented generation and fine tuning for Text-to-SQL*. *arXiv.* https://doi.org/10.48550/arXiv.2404.12560

Volvovsky, S., Marcassa, M., & Panbiharwala, M. (2024). DFIN-SQL: Integrating focused schema with DIN-SQL for superior accuracy in large-scale databases. *arXiv.* https://doi.org/10.48550/arXiv.2403.00872

Wang, B., Shin, R., Liu, X., Polozov, O., & Richardson, M. (2021). RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. *arXiv.* https://doi.org/10.48550/arXiv.1911.04942

Wang, J., Wen, J. R., Lochovsky, F., & Ma, W. Y. (2004, August). Instance-based schema matching for web databases by domain-specific query probing. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases* (Vol. 30, pp. 408–419). Toronto, Canada.

Wang, Y., Liu, P., & Yang, X. (2025). *LinkAlign: Scalable schema linking for real-world large-scale multi-database text-to-SQL*. *arXiv.* https://doi.org/10.48550/arXiv.2503.18596

Wang, Z., Yang, J., & Ye, X. (2020, November). Knowledge graph alignment with entity-pair embedding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1672–1680). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.130

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35, 24824–24837.

Wu, L., Petroni, F., Josifoski, M., Riedel, S., & Zettlemoyer, L. (2019). Scalable zero-shot entity linking with dense entity retrieval. *arXiv.* https://doi.org/10.48550/arXiv.1911.03814

Xia, C. S., Deng, Y., Dunn, S., & Zhang, L. (2024). Agentless: Demystifying LLM-based software engineering agents. *arXiv.* https://doi.org/10.48550/arXiv.2407.01489

Xu, X., Liu, C., & Song, D. (2017). SQLNet: Generating structured queries from natural language without reinforcement learning. *arXiv.* https://doi.org/10.48550/arXiv.1711.04436

Yaghmazadeh, N., Wang, Y., Dillig, I., & Dillig, T. (2017). SQLizer: Query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, *1*(OOPSLA), 1–26. https://doi.org/10.1145/3133887

Yang, Y., Chen, M., & Gao, B. (2008, November). An effective content-based schema matching algorithm. In *2008 International Seminar on Future Information Technology and Management Engineering* (pp. 7–11). IEEE. https://doi.org/10.1109/FITME.2008.38

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., & Radev, D. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. *arXiv.* https://doi.org/10.48550/arXiv.1809.08887

Zapilko, B., Zloch, M., & Schaible, J. (2012, November). Utilizing regular expressions for instance-based schema matching. *In OM*.

Zha, L., Zhou, J., Li, L., Wang, R., Huang, Q., Yang, S., Yuan, J., Su, C., Li, X., Su, A., Zhang, T., Zhou, C., Shou, K., Wang, M., Zhu, W., Lu, G., Ye, C., Ye, Y., Ye, W., Zhang, Y., Deng, X., Xu, J., Wang, H., Chen, G., & Zhao, J. (2023). TableGPT: Towards unifying tables, natural language and commands into one GPT. *arXiv.* https://doi.org/10.48550/arXiv.2307.08674

Zhai, B., Xu, C., He, Y., & Yao, Z. (2025). ExCoT: Optimizing reasoning for Text-to-SQL with execution feedback. *arXiv.* https://doi.org/10.48550/arXiv.2503.19988

Zhang, H., Cao, R., Chen, L., Xu, H., & Yu, K. (2023). ACT-SQL: In-context learning for text-to-SQL with automatically-generated chain-of-thought. *arXiv.* https://doi.org/10.48550/arXiv.2310.17342

Zhang, H., Dong, Y., Xiao, C., & Oyamada, M. (2023). Jellyfish: A large language model for data preprocessing. *arXiv.* https://doi.org/10.48550/arXiv.2312.01678

Zhang, H., Dong, Y., Xiao, C., & Oyamada, M. (2023). Large language models as data preprocessors. *arXiv.* https://doi.org/10.48550/arXiv.2308.16361

Zhang, J., Shin, B., Choi, J. D., & Ho, J. C. (2021). SMAT: An attention-based deep learning solution to the automation of schema matching. In L. Bellatreche, M. Dumas, P. Karras, & R. Matulevičius (Eds.), *Advances in Databases and Information Systems: 25th European Conference, ADBIS 2021, Tartu, Estonia, August 24–26, 2021, Proceedings* (Vol. 12843, pp. 260–274). Springer. https://doi.org/10.1007/978-3-030-82472-3_19

Zhang, M., Hadjieleftheriou, M., Ooi, B. C., Procopiuc, C. M., & Srivastava, D. (2011, June). Automatic discovery of attributes in relational databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (pp. 109–120). https://doi.org/10.1145/1989323.1989336

Zhang, M., Hadjieleftheriou, M., Ooi, B. C., Procopiuc, C. M., & Srivastava, D. (2010). On multi-column foreign key discovery. In *Proceedings of the VLDB Endowment, 3*(1–2), 805–814. https://doi.org/10.14778/1920841.1920944

Zhang, W., Wang, Y., Song, Y., Wei, V. J., Tian, Y., Qi, Y., Chang, J. H., Wong, R. C., & Yang, H. (2024). Natural language interfaces for tabular data querying and visualization: A survey. In *IEEE Transactions on Knowledge and Data Engineering, 36*(11), 6699–6718. https://doi.org/10.1109/TKDE.2024.3400824

Zhili Shen, Pavlos Vougiouklis, Chenxin Diao, Kaustubh Vyas, Yuanyi Ji, & Jeff Z. Pan (2024, November 4). Improving Retrieval-augmented Text-to-SQL with AST-based Ranking and Schema Pruning. *arXiv.* https://doi.org/10.48550/arXiv.2407.03227

Zhong, V., Xiong, C., & Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv.* https://doi.org/10.48550/arXiv.1709.00103

Zhou, S., Rijhwani, S., & Neubig, G. (2019). Towards zero-resource cross-lingual entity linking. *arXiv.* https://doi.org/10.48550/arXiv.1909.13180

10 Appendix A

10.1 Starting Prompt for LLM Question-Answering Phase

Answer the question based on the following data columns.

Table: <database>.<schema>.<table_id>

Table Description: <Optional table description if available>

Column 1: <column_name_1>

Description: <Optional column description>

Type: <Optional column type>

Sample Values: <Optional sample values (truncated if too long)>

Column 2: <column_name_2>

Description: ...

Type: ...

Sample Values: ...

... (repeat for more columns and tables)

Question: <Natural language question provided by the user>

Expected Query Output:

Your query must return exactly the following columns in this order:

| "<column_1>" | "<column_2>" | ... |

Instructions:

- These are the **column names in the final output of the query**.

- Use aliases in the SELECT clause if needed to match this output format.

- Do not include any additional columns.

- The SELECT clause must produce exactly these column names and in this order.

Generate a valid and efficient SQL query in Snowflake SQL dialect that answers the question above.

- Use correct table and column references based on the schema and table details provided. The FROM clause must always be in the format: database.schema.table, copy this exactly from the table definitions above.

- When you need to calculate or use conditions based on an unknown parameter, don't set it to a value but use a calculation (e.g., long/medium/short durations based on midpoints between min/avg/max).

- Column names in the SELECT clause must be enclosed in double quotes (e.g., "column_name") and in lowercase. If that fails, try uppercase or Capitalized.

- Do not assume data or filters unless explicitly provided.

- For relative date filters like 'one year ago', use:

`WHERE refresh_date BETWEEN DATEADD(year, -1, CURRENT_DATE()) AND CURRENT_DATE()`

- Avoid non-SQL functions like ARRAY_EXISTS or LLM-specific syntax.

USE SNOWFLAKE SQL DIALECT ONLY.

Wrap the SQL query between the tags `### SQL_QUERY_START` and `### SQL_QUERY_END`.

Example Output:

SQL_QUERY_START

SELECT ...

FROM ...

WHERE ...

GROUP BY ...

ORDER BY ...

SQL_QUERY_END

10.2 Prompt Generator Assistant

You are a Snowflake SQL query generator. Your job is to generate a valid and efficient query using the **Snowflake SQL dialect** only.

Instructions:

- You will receive a natural language question and detailed database schema information.

- Your task is to generate a SQL query that accurately answers the question using only the provided schema.

Strict requirements:

1. Use only **Snowflake SQL dialect** — no functions or syntax from other dialects (e.g., no `LIMIT` without `FETCH`, no `TEXT[]`, no `ARRAY`, etc.).

2. The FROM clause must have three parts: `database.schema.table`. This information is given in the information provided as TABLE:

4. Do not assume columns or data unless explicitly described in the schema.

5. If using date filters like “one year ago”, use Snowflake syntax:

`WHERE refresh_date BETWEEN DATEADD(year, -1, CURRENT_DATE()) AND CURRENT_DATE()`

6. Avoid hardcoded values when the question calls for reasoning (e.g., use calculations like midpoint/average for ranges).

7. Never use non-SQL logic or LLM-specific language (e.g., no `ARRAY_EXISTS`, `filter()`, or pseudo-code).

8. Use aliases and readable formatting when appropriate.

9. If the question asks for the id and you also have a column with the name, provide both of them in the SELECT statement (HAVING MORE COLUMNS IS BETTER THAN HAVING LESS COLUMNS).

10. MAKE SURE THAT THE COLUMNS NAMES ARE EXACTLY AS THEY ARE IN THE INFORMATION PROVIDED, IF THEY ARE IN LOWERCASE, WRAP THEM IN DOUBLE QUOTES (i.e. "column_name" instead of column_name).

11. DO NOT USE TO_TIMESTAMP_NTZ("column_name" / 1000000) BUT USE TO_TIMESTAMP("column_name" / 1000000)

MAKE SURE TO FOCUS ON THE CHANGES YOU ARE TOLD TO MAKE !

THE QUERIES NEED TO BE AS ROBUST AS POSSIBLE

Format:

- Wrap the final SQL query between these tags:

SQL_QUERY_START

SELECT ...

FROM database.schema.table

WHERE ...

SQL_QUERY_END

Goal:

Generate only one correct, executable Snowflake SQL query that directly answers the question using the provided schema, and that perfectly matches

Only return the SQL query between the tags. Do not include explanations, comments, or the question itself in the response.

10.3 Prompt Checker Assistant

You are simulating a user interacting with an LLM to generate a Snowflake SQL query that answers a natural language question.

Your task is to evaluate the generated query and provide structured feedback. Follow this process:

---

1. Assess Relevance:

- Does the query logically and semantically answer the user's original question?

- Does it use appropriate columns, tables, and filters?

2. Check Syntax:

- Is the query valid Snowflake SQL?

- If there are syntax errors, identify them precisely.

- If the problem is with an INVALID IDENTIFIER, IT MOST LIKELY MEANS THAT THE COLUMN NAME WAS NOT REFERENCED CORRECTLY SO CHECK EACH COLUMN AND MAKE SURE THAT THEY ARE EXACTLY (EVEN THE CASE) AS THEY ARE IN THE INFORMATION PROVIDED.

A LOT OF TIMES THE INVALID QUALIFIER IS DUE TO THE COLUMN NAME NOT BEING IN BETWEEN DOUBLE QUOTES (e.g. "column_name" instead of column_name).

- THE QUERIES NEED TO BE AS ROBUST AS POSSIBLE

---

3. Evaluate the Output (if present):

- Does the result make sense given the question?

- If the output is **null, empty, or full of zeros**, you must **never** assume the query is correct.

- Investigate what caused the empty or meaningless result. This could be due to:

  - Missing or incorrect joins.

  - Overly restrictive or wrong WHERE clauses.

  - Invalid column references or filter logic.

  - Improper aggregations or GROUP BY conditions.

- Remember: **there should always be a result**, unless the question explicitly calls for empty or filtered cases.

---

4. Provide Feedback:

Write a clear, concise paragraph to send back to the LLM. It should:

- Explain any issues in logic or syntax.

- Suggest how to fix them (clearly and specifically).

- If there are invalid identifiers, use the provided valid identifiers to correct the query.

- If a fix is obvious, write the revised query and return it as described below.

- If the query matches the user's intent **but returns no or zero results**, you must explain what may be causing this and request a fix.

---

5. Format for Output (main logic queries):

If you are **confident** you can fix the original query and generate a correct one, output only the query in this format:

SQL_QUERY_START

SELECT ...

FROM database.schema.table

WHERE ...

SQL_QUERY_END

⚠️ Important:

- Only generate a new query if the previous step did **not** already include one.

- Do **not** generate speculative or placeholder queries.

- Do **not** assume the model has access to runtime data—only evaluate queries and outputs as given.

---

6. Validate Output Columns:

- If the expected output column names are provided, ensure the query returns **only** those columns, in the correct order and format.

- Do not allow extra columns or missing columns.

- If needed, use column aliases (e.g., `AS "city_one"`) to match expected output exactly.

- If the output columns are incorrect, request that the query be rewritten with the correct `SELECT` clause.

---

7. Final Judgment:

If the query is **correct** and the **output is not empty**, and both align with the user's question:

- Respond only with: `Query validated. Marking as final.`

If the query looks syntactically fine but the **output is empty**, that is **not sufficient** to mark it as final. You must identify what in the query might have led to this, or provide a standalone query to assist.

10.4 System Prompt for Table Description Inference

10.4.1 Grouped Tables

You are generating metadata for a **grouped database table template**.

Each table in the group follows the same schema but varies by year, region, or other dimensions.

Your task is to describe what kind of data is captured across the group — not for a single instance.

Instructions:

- Write a concise 1–2 sentence **description** of what this group of tables contains, based on the database name, table name, column names, types, description and sample row.
- Then provide at least 10 **keywords or phrases** users might use to search for tables like this.
- Focus on concepts, entities, or analysis that the data enables. Avoid file names or exact column names.

Output format:

"{\n"

"description": "...",

"keywords": ["...", "...", ...]

"}"

10.4.2 Ungrouped tables

"You are generating metadata for a **single database table**.\n"

"This table is not part of a larger group or template.\n"

"Your task is to describe exactly what this table contains and how users might refer to it, based on the database name, table name, column names, types, descriptions and sample row\n\n"

"Instructions:\n"

"- Write a concise 1–2 sentence **description** of what this table contains.\n"

"- Then provide at least 10 **keywords or phrases** users might use to search for this table.\n"

"- Focus on meaningful terms and real-world concepts. Avoid repeating column names unless standard terminology.\n\n"

"**Output format**:\n"

"{\n"

' "description": "...",\n'

' "keywords": ["...", "...", ...]\n'

"}"

10.5 Prompt for Column Description Inference

10.5.1 System Prompt

You are generating metadata for **multiple columns** in a database table.
Your task is to describe what each column contains and how users might refer to it, based on the table's description and other columns.
For each column:
- Write a concise, clear sentence describing its **meaning and purpose**.
- Then provide **10 relevant keywords or phrases** capturing its semantic content (used for semantic search and matching).
- Focus on real-world concepts. Avoid redundancy or just repeating the column name.
Output format:

```
{
  "column_name_1": {
    "description": "...",
    "keywords": ["...", "..."]
  },
  "column_name_2": {
    ...
  }
}
```

10.5.2 User Prompt Example

You are working with columns from the `patient_records` table, which belongs to `health_data` in the `clinical_db` database.

- Table Description: This table stores patient visits and medical histories.

Note: Only a subset of the table's columns is provided below.

- Existing Columns:
  - visit_date: date of patient's visit
  - diagnosis_code: code used to classify the patient's condition
- Target Columns (to describe):
  - blood_pressure (string)
  - medication (string)

10.6 Prompt for Paraphrasing

You are a natural language assistant that reformulates analytical questions for use in a retrieval-augmented SQL generation system.
Your task is to take a user's natural language question and generate 1 different reformulations that:
- Use varied phrasing and sentence structure
- Preserve the precise analytical intent of the original question
- Clearly express the entities, time logic, and conditions involved
- Are suitable for matching with table and column descriptions in a vector database
Each reformulation should be accurate, well-structured, and reflect the same logical meaning as the original using synonyms and restructuring the question completely are allowed (e.g. completed orders could be completed or delivered orders...)
NO TWO REFORMULATIONS SHOULD BE TOO SIMILAR. Each should be distinct in wording and structure while maintaining the original intent.

THE REFORMULATIONS SHOULD HAVE DIFFERENT WORDING AS MUCH AS POSSIBLE.

Return the output in the following JSON format:

```
{
"intent_paraphrases": [
  "Reformulation 1",
  ...
  "Reformulation 3"
]
}
```

10.7 Prompt for Entities and Keywords for NL Question

You are an AI assistant designed to extract relevant entities, synonyms, metrics, time expressions, conditions, and other concepts from a user's natural language question. These elements will be used for hybrid retrieval in a Qdrant vector database to match relevant database metadata (such as tables and columns) and assist with SQL query generation.

Please analyze the user's questions and extract the following:

1. **Entities**: General schema-relevant concepts implied in the question, such as "location", "date", "timestamp", "category", "score", etc should also be included (e.g. "last year" or "in the past 5 days" should result in "date" or "timestamp"). If any time-expressions are present, give them the label "date" or "timestamp".

2. **Synonyms**: Alternate or related terms that a user might use for the identified entities or other key concepts, which could help match to schema elements.

3. **Metrics**: Any quantitative concepts or aggregation methods mentioned, expressed as full phrases (e.g., "highest search score", "total count").

4. **Time Expressions**: Literal time-based phrases from the question (e.g., "one year ago", "last month", "from 2020 to 2023").

5. **Conditions**: Any filters, comparisons, or logical operations implied by the question (e.g., "greater than 100", "for California only").

6. **Other Concepts**: Additional meaningful terms or domain-specific ideas that help clarify the user's intent (e.g., "top rising terms", "trending topics").

INPUT: A list of natural language questions, including the original and its reformulations.

Return your output in this JSON format:

```
{
"entities": [...],
"synonyms": [...],
"metrics": [...],
"time_expressions": [...],
"conditions": [...],
"other_keywords": [...]
}
```

5.3.2

10.8 Example Query of Providing Distinct Values to the LLM

The issue with this is that even if I identify the columns where it states the condition of "helmets/no helmets", the LLM wouldn't know on which values to condition precisely. This is the query outputted without the sample values:

```
SELECT
    CASE
        WHEN COUNT(*) = 0 THEN 0
        ELSE (SUM(CASE WHEN LOWER("victim_safety_equipment_1") = 'helmet' THEN 1 ELSE 0 END) *
                                                                    100.0) / COUNT(*)
    END AS "percent_killed_helmet_used",
    CASE
        WHEN COUNT(*) = 0 THEN 0
        ELSE (SUM(CASE WHEN LOWER("victim_safety_equipment_1") <> 'helmet' OR
                "victim_safety_equipment_1" IS NULL THEN 1 ELSE 0 END) * 100.0) / COUNT(*)
```

```
    END AS "percent_killed_helmet_not_used"
FROM CALIFORNIA_TRAFFIC_COLLISION.CALIFORNIA_TRAFFIC_COLLISION.VICTIMS
WHERE "victim_role" = 'motorcyclist'
  AND "victim_degree_of_injury" = 'fatal';
```

We can see here that it takes the *victim_safery_equipment_1 = "helmet"* but there is no value with this condition. What we are actually looking for is a condition on "driver, motorcycle helmet not used", "driver, motorcycle helmet used", "passenger, motorcycle helmet used" and "passenger, motorcycle helmet not used". Therefore, in many cases, providing all the distinct sample values gives the LLM more context. For the same question as before, the output would then change to:

```
WITH fatal_motorcycle_victims AS (
    SELECT
        CASE
            WHEN LOWER("victim_safety_equipment_1") IN ('driver, motorcycle helmet used', 'passenger,
                 motorcycle helmet used')
              OR LOWER("victim_safety_equipment_2") IN ('driver, motorcycle helmet used', 'passenger,
                 motorcycle helmet used')
            THEN 'helmet_used'
            WHEN LOWER("victim_safety_equipment_1") IN ('driver, motorcycle helmet not used', 'passenger,
                 motorcycle helmet not used')
              OR LOWER("victim_safety_equipment_2") IN ('driver, motorcycle helmet not used', 'passenger,
                 motorcycle helmet not used')
            THEN 'helmet_not_used'
            ELSE 'unknown'
        END AS helmet_status
    FROM CALIFORNIA_TRAFFIC_COLLISION.CALIFORNIA_TRAFFIC_COLLISION.VICTIMS
    WHERE "victim_role"            IN ('driver','passenger')
      AND "victim_degree_of_injury" = 'killed'
)
SELECT
    ROUND(
        100.0 *
        SUM(CASE WHEN helmet_status = 'helmet_used' THEN 1 END) / NULLIF(COUNT(*),0), 2) AS
            percent_killed_helmet_used,
    ROUND(
        100.0 *
        SUM(CASE WHEN helmet_status = 'helmet_not_used' THEN 1 END)/ NULLIF(COUNT(*),0), 2) AS
            percent_killed_helmet_not_used
FROM fatal_motorcycle_victims;
```