POLITECNICO DI TORINO

Master's Degree in DATA SCIENCE AND ENGINEERING

Master's Degree Thesis

Enhancing User-Database Interaction Through a User-Friendly Platform Leveraging LLMs

Supervisors
Prof. LUCA CAGLIERO

Company Supervisor
Ing. EMANUELE MUNAFÒ

Candidate
MARCO CASTIGLIA

April 2025

Summary

This thesis explores the most suitable language models for improving the interaction between users and databases via natural language, reducing or completely neglecting the need for technical knowledge. We address the increasing complexity of databases and the demand for intuitive query interfaces by developing a proof-of-concept platform that transforms natural language queries into SQL-like statements for efficient data retrieval. The system supports multiple databases and enables users to analyze query results in various formats. Key contributions include benchmarking state-of-the-art models, exploring novel prompt enrichment approaches like Chain-of-Thought reasoning and additional data samples, designing a comprehensive system architecture, and conducting experiments to evaluate effectiveness. The experimental results confirm that the aforementioned techniques improve accuracy over zero-shot cases at a slight increase in cost and processing time, demonstrating their potential for practical applications considering proper trade-offs. Our work enhances database accessibility and usability, enabling users to interact intuitively with complex data structures and improving data-driven decision-making processes.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context and Motivation

In recent years, Natural Language Processing, a branch of Artificial Intelligence, has experienced significant progress, driven by the development of sophisticated algorithms, large language models, and the availability of extensive datasets. These advancements have substantially improved the ability of machines to understand, interpret, and generate human language. NLP comprises a broad range of tasks, which can be grouped into several core areas, such as text classification, machine translation, text summarization, speech recognition, and text generation, among others.

While numerous NLP tasks are worthy of discussion, this thesis focuses specifically on the text generation task, as it is central to the scope of our research. Text generation is particularly noteworthy due to its broad range of applications, including chatbots, virtual assistants, automated content creation, and data retrieval systems. These applications present significant opportunities for innovation, but also pose various challenges, both in terms of research and real-world deployment.

While NLP is a dynamic and evolving sector characterized by its ability to handle unstructured and ambiguous human language through sophisticated algorithms and large language models, databases operate in a more deterministic and structured environment. Databases are built upon well-defined schemas and rely on precise query languages like SQL (Structured Query Language) to retrieve and manipulate data. Text-to-SQL bridges these two sectors by translating the flexibility of natural language into the rigid structure of SQL queries. This task allows users to interact with databases

using conversational language, making complex data systems more accessible to non-experts.

In this thesis, we focus on traditional relational databases and the task of transforming natural language text into SQL queries for generic relational databases. This field of research is relatively recent, gaining significant hype over the past decade due to breakthroughs in deep learning technologies.

1.2 Problem Statement and Challenges

Recently, the field of Text-to-SQL has increasingly turned to machine learning methods and large language models to improve the accuracy and robustness of SQL query generation. Early approaches primarily relied on sequence-to-sequence [1] models and Long Short-Term Memory [2] networks, which were some of the first neural architectures applied to this task. However, transformer-based models have now become the standard due to their superior ability to capture complex patterns in natural language and effectively translate them into structured SQL queries. Large language models, such as BERT [3], T5[4], and GPT variants [5] [6], have shown exceptional promise in understanding the nuances of natural language and generating syntactically correct SQL queries. Recently, even more advanced models have been published, further pushing the boundaries of what is possible in this domain.

Despite these advancements, user-database interaction remains a complex and evolving process, requiring careful optimization to balance performance, security, and scalability while ensuring a seamless user experience. To make a more practical example in an enterprise context, an employee with a technical background has the skills to interact with a database. However, people with less technical roles, such as stakeholders, project managers, or marketing team members, may not know the exact commands to extract the necessary information. This information may be crucial for strategic decisions like marketing campaigns or new product directions.

Then, this domain has many challenges, which can be broadly categorized into two areas: challenges specific to the Text-to-SQL task itself, i.e., the generation of text, and challenges associated with building robust and helpful Text-to-SQL systems or platforms to bridge the gap between database querying and, especially, non-experts. These include issues related to understanding and generating SQL queries from natural language input, as well as the complexities involved in creating end-to-end systems that can operate

effectively in real-world scenarios.

Typical challenges of the task itself are the same related to the modern approaches based on deep neural networks and large language models, therefore, for instance, hallucinations, since these are probabilistic models and the output is just a prediction based on the given prompt, or difficulties in capturing long context windows. Another significant challenge is the cost of inference and/or fine-tuning or training, both in terms of resources and expenses, if we opt for proprietary models that require paid access through API keys.

Then, of course, the obvious challenge is to generate an accurate query in SQL language. The latter requires to follow a precise syntax, making even minor errors potentially disruptive. It demands strict adherence to predefined rules, including correct keyword usage, proper table and column references, and logical query structuring. Additionally, an optimal SQL query should not just be correct but also efficient. Poorly written queries can lead to significant performance bottlenecks. The challenges of building end-to-end Text-to-SQL systems are equally significant. One particularly challenging aspect is handling natural language variability, which may be ambiguous and is usually context-dependent. Different users may phrase the same query in multiple ways, using different synonyms and sentence structures, or even writing grammatical errors. This variability complicates the task of generating complex SQL queries, especially those that require multiple joins and nested sub-queries, as the model must accurately interpret and adapt to the complexity of each query.

Furthermore, in real-world scenarios, databases do not always follow best practices. They are often much larger and more complex than small test datasets, making query formulation and optimization even more challenging. Structural inconsistencies, poor schema design, and non-intuitive table or column naming conventions can introduce additional difficulties. Therefore, capturing the intent of these diverse expressions and linking them to a fixed database schema is another challenge, necessitating the development of various specialized approaches.

Lastly, while real-time performance is not strictly necessary for this kind of systems, they should still be fast and efficient. Achieving this performance while maintaining high accuracy and scalability across various databases and user queries remains a challenge.

1.3 Objectives and Contribution

The challenges and considerations discussed in the previous section have been explored in various studies, with several papers conducting benchmarks that evaluate models using a range of metrics and perspectives. The primary focus of this thesis will be on the insights provided by the DAIL-SQL paper [7], which achieved state-of-the-art results on the Spider benchmark dataset using mainly proprietary models tailored for the Text-to-SQL task, as well as some open-source models. This paper provides a comprehensive analysis by comparing different models proposing different prompt engineering strategies, including in-context learning, few-shot learning, and other methods that will be discussed in detail later.

Building upon the foundational research outlined in previous studies, this thesis, developed in collaboration with Connect IT Reply, explores the most suitable language models for improving the interaction between users and databases via natural language, reducing or completely neglecting the need for technical knowledge. The project extends the results of the paper on DAIL-SQL by evaluating new models in the best-case scenarios analyzed in the paper itself. Furthermore, a proof of concept was developed to demonstrate the feasibility, utility, advantages, and limitations of the proposed solution.

The contributions of this work can be broadly categorized into four main areas:

- **Model Evaluation in 0-Shot, 5-Shot, and 9-Shot Learning:** We assess the performance of various large language models on the Spider dataset by providing contextual examples of other questions and queries from the training set. The evaluated models include GPT-3.5 Turbo, GPT-4o Mini, Gemini 1.5 Flash, and Gemini 1.5 Pro.

- **Enhanced Prompting with `INSERT` Statements:** To improve the models' understanding and accuracy, we introduce examples of `INSERT` statements within the prompts. This includes explicit examples of table entries, providing richer context to the models.

- **Chain-of-Thought (CoT) Prompting:** We design and test prompts inspired by the Chain-of-Thought reasoning approach, aiming to enhance logical inference and structured query generation.

- **Platform Proof of Concept (PoC):** We develop a platform to enhance user-database interactions. This platform is structured into four main sections, enabling users to manage various databases, generate and execute queries, and consult query results in multiple formats.

Additionally, the thesis discusses several minor contributions, including the implementation of a basic Retrieval-Augmented Generation (RAG) pipeline and the evaluation of small, open-source models to explore the feasibility of locally hosting a model, relying solely on prompt engineering techniques.

1.4 Brief Overview

The following is a brief overview which serves as a roadmap to guide the reader throughout the thesis.

Chapters 2 and 3 delve into the background knowledge, related works, and state-of-the-art advancements in the NLP and Text-to-SQL fields. The former provides a historical perspective on natural language processing, explores existing database technologies, and discusses the main large language models and tools of generative AI. Furthermore, Chapter 3 offers a comprehensive overview of the Text-to-SQL task, highlighting recent developments and key research directions, in particular dealing with the latest papers of this sector.

Chapter 4 focuses on the experimental setup, which leads to the evaluation of various proprietary models and techniques. It introduces the datasets used for evaluation, outlines the relevant metrics, and presents a detailed review of the models under consideration, and the single experiments.

Chapters 5 and 6 provide a comprehensive analysis of the results, comparing the performance of different models and approaches against established baselines, and the discussion of these findings identifying areas for future research and leading to the enterprise use case and proposed solution.

Chapter 7 shifts the focus to the proposed solution for implementing the proof-of-concept platform. This chapter provides a detailed description of the selected model and the strategies employed to generate SQL queries. It outlines the specific requirements and objectives of the intended use case, discusses the real-time data retrieval mechanism, and addresses system integration and potential future developments.

Chapter 8 concludes the thesis, summarizing key contributions, highlighting the significance of the research, and providing a final outlook on potential future directions.

Chapter 2

Preliminaries

This chapter aims to provide a concise yet comprehensive overview of the foundational concepts necessary to thoroughly explore the topics, objectives, and solutions presented in this thesis.

2.1 Natural Language Processing

This branch of Artificial Intelligence has evolved significantly since its inception in the 1950s. The field has seen several paradigm shifts, from rule-based systems to statistical methods and, more recently, to deep learning approaches. Over time, these advancements have improved machines' ability to understand and interpret human language. Natural Language Processing tasks can be divided into several macro-areas based on their specialization.

Text classification Categorizing and identifying specific elements in the text based on the desired task.

Text generation Creating text from a given input, such as translation or summarization.

Information retrieval Extracting information from text to provide answers.

Speech modeling Recognizing, generating, and understanding speech in various applications.

This thesis is mainly focused on text generation, particularly Text-to-SQL. Therefore, the text generated will be an SQL statement to analyze and query an actual database. For this reason, the methods we will deal with are related to the advancements in this application area.

2.1.1 Rule-based models

The initial approaches emerged from an interest in machine translation. They were based on hard-coded rules, the goal of which was to find patterns in the text and provide an output accordingly. Despite their application in specific fields and elementary tasks nowadays, they have many limitations. They must be adapted for different domains and subtasks, limiting their flexibility and scalability and can work only with very precise structures and patterns in the text. Therefore, they struggle to capture the complexity of human language.

An example of these systems is ELIZA [8], a chatbot developed in the mid-1960s at MIT to replicate a conversation between a patient and a psychotherapist. It simply operates by rephrasing input text as a question, similar to how a psychologist would respond. Although these methods are basic, they can still be combined with more advanced approaches, as the rules created are not static but are applied, processed, and iteratively refined.

2.1.2 Statistical Models, RNNs, and LSTMs

In the 1980s, the development of statistical models changed the perspective and boosted the research in NLP after a slowdown. The idea is to predict and output a word or a sentence according to probabilities. For instance, the count-based approach involves calculating the frequency of a particular word given the previous words. Significant models are Naive Bayes and N-grams [9], where the parameter N is related to the number of words considered to calculate the probability. The underlying theory is based on conditional probability and Bayes' theorem.

Statistical models struggled with long-range dependencies between words; however, they introduced the community to probabilistic thinking in language modeling. Neural networks brought a huge and significant shift from feature engineering, typical of rule-based and statistical models, to end-to-end learning. The link between input and output is no longer linear.

The inception of recurrent neural networks (RNNs) [10] [11] in 1985 was

groundbreaking. This kind of architecture is peculiarly able to handle variable-length sentences, keeping much more context than previous approaches. The key innovation lies in the network's recurrent connections, where the output of one time step is fed as input to the next, thereby maintaining a form of memory across time steps, as illustrated in the figure.

However, despite this architecture's advantages, RNNs face challenges during backpropagation. The issue of vanishing and exploding gradients restrains the model's ability to learn effectively over extended sequences, leading to suboptimal performance. Therefore, the context over long time windows is limited, restricting the model's overall effectiveness in handling complex linguistic patterns.

To address the limitations of traditional RNNs, Hochreiter and Schmidhuber introduced the Long Short-Term Memory (LSTM) [2] architecture in 1997. LSTMs are a specific type of RNN that incorporates "gates" to control the flow of information, allowing them to learn long-range dependencies more effectively. These gates include input, output, and forget gates, allowing the model to retain and transfer information across different layers, mitigating the vanishing/exploding gradients effect. LSTMs are used as memory cells, or blocks, in more complex architectures. On the one hand, this innovative approach marked a significant advancement in sequential modelling, outperforming traditional RNNs and becoming a popular choice for NLP applications. On the other hand, it is computationally costly due to the complex gating mechanisms.

2.1.3 Word Embeddings

As we work in the field of natural language processing, the input data for a model are obviously words. However, a machine cannot directly interpret words, so they are transformed into vectors of numbers. Traditionally, words were converted using one-hot encoding or similar methods, creating sparse high-dimensional vectors. Therefore, words are considered simply as a group of letters without capturing their relationship with other words.

In 2013, the publication of "Efficient Estimation of Word Representations in Vector Space" [12] revolutionized the way input data could be fed to neural networks.

The core idea is to represent words as vectors in a continuous vector space, where semantically similar words are close to each other. This mechanism is based on neural networks that learn the embeddings after training over text datasets. The NN learns semantic and syntactic relationships between

words, allowing for more accurate and meaningful language modeling. Also, this approach reduces dimensionality compared to previous approaches. Nonetheless, it is essential to note that word embeddings can only represent words encountered during training and may still face challenges when words have multiple meanings.

Word2Vec (2013) [12], GloVe (2014) [13] , and FastText (2018) [14] are pretty common approaches for computing and learning word embeddings.

2.1.4 Sequence to sequence models

This section is devoted to the so-called Seq2Seq [1] models. These models combine the strengths of LSTM networks and word embeddings, integrating them into a novel architecture known as encoder-decoder. As can easily be guessed, the architecture has two blocks from a macroscopic point of view. The encoder processes and compresses the input sequence into a fixed-length vector representation, capturing its semantic and contextual information. The decoder then utilizes this representation to generate an output.

Apart from the advantages of employing LSTMs, we can highlight two main advantages of this approach. Firstly, since the encoder, in practice, summarizes the context of the input sequence, context retention is enhanced. Secondly, it is possible to process input sequences of varying lengths and generate output sequences of different lengths, making them versatile for tasks like machine translation or text summarization where input and output lengths may differ significantly.

2.1.5 Transformer Architectures

All the above-mentioned methods and networks process data sequentially, making the training computationally expensive and time-consuming. Vaswani et al., in 2017, came up with a novel approach in their paper "Attention is all you need" [15], where the new proposed architecture is totally built on the new concept of attention.

Transformers still present an encoder-decoder structure, similar to Seq2Seq, but have their innovation in the Multi-head Self-Attention layers. Their role is to weigh the importance of a word in a sequence, which means that the model can now focus on multiple parts of a sentence simultaneously. Therefore, thanks to the structure of this neural network, as shown in the figure, input data processing is now parallelized. This eliminates the need for recursion, such as in RNNs, thus making models more computationally efficient and

performing. Parallelization is obtained because the model processes the entire sequence at once, making it easy to parallelize the training. For this reason, it was necessary to incorporate information about the position of a specific word in the sentence. So, there were not only word embeddings but also positional embeddings. This mechanism is not limited to text; it can be used for a variety of applications.

Some of the first and most important language models based on transformer architectures are BERT [3] and GPT [5] [6]. Nowadays, these represent the standard for large language models, which will be discussed in the next section.

The idea behind both models is the pre-training on large corpora of unlabelled text data, so that they will be task-agnostic after learning a general distribution of words. After this, both can be fine-tuned to achieve state-of-the-art performances in task-specific benchmarks. Therefore, these models aim to exploit the exceptional availability of large text datasets using unsupervised learning techniques. None of the two prevails over the other, it is just a matter of which task one aims to address.

As regards GPT, the acronym stands for Generative Pre-Trained Transformer. It is an autoregressive model based on the transformer's decoder block and uses a left-to-right approach: given a sequence, it predicts the next word based on the preceding words. Hence, the processing is unidirectional. Due to this characteristic, GPT is particularly strong in generative tasks like text generation.

On the contrary, BERT is based on the encoder part of a transformer. The acronym, in fact, stands for Bidirectional Encoder Representations from Transformers. Here, the model predicts masked tokens according to both left and right dependencies in the sentence, capturing a richer context. Thanks to this distinctive trait, BERT is more predisposed to understanding tasks, such as summarization and question answering.

Beyond BERT and GPT, two other transformer-based architectures that have significantly influenced the field of NLP are T5 (Text-to-Text Transfer Transformer) [4] and BART (Bidirectional and Auto-Regressive Transformer) [16]. Both models adopt an encoder-decoder structure, making them particularly suited for tasks requiring both understanding and generation of text.

T5, developed by Google, treats every NLP task as a text-to-text problem, meaning that all inputs and outputs are formatted as text. A distinctive feature of this model is that the task is specified within the input prompt, allowing it to generalize across different domains efficiently.

On the other hand, BART, introduced by Facebook AI (Meta), is designed as a denoising autoencoder. It is trained by randomly masking words or shuffling sentence order and then learning to reconstruct the original version. This pre-training approach makes BART highly effective for text generation and correction tasks. Unlike GPT, which is only decoder-based, and BERT, which is only encoder-based, BART combines both mechanisms, benefiting from bidirectional context understanding while maintaining strong generative capabilities.

These models, along with BERT and GPT, represent essential components of modern Large Language Models.

2.1.6 Large Language Models

The development of the above-mentioned models marked a breakthrough in natural language processing, becoming the foundation for what would come to be known as Large Language Models (LLMs). These models exhibited outstanding performance across various tasks, from text generation to semantic understanding. The advancements gave the NLP community a clearer understanding of how to leverage large-scale unsupervised pre-training, combined with fine-tuning, to achieve state-of-the-art results. They can already be considered "large" models, even if they have "only" a few hundred million parameters, a very small number compared to current models. Further research, which is still ongoing nowadays, is leading to the development of LLMs characterized by their exponentially larger scale, both in terms of model parameters and volume of data used for training. This has been possible because technologies and resources to support such training have increased enormously over the years.

In the following section, we explore the rise of LLMs, examining their structure, training paradigms, and their impact on AI-driven applications today and in the future. We will then discuss open-source and proprietary models that researchers, developers, and almost everyone who has access to the Internet use today because of their impressive state-of-the-art performances.

Training and Learning Paradigms

Training LLMs is an articulated process that may include several stages, including not only pre-training on large text corpora but also fine-tuning to specialize them in specific tasks and using advanced prompt engineering techniques to guide their responses. During pre-training, the model is exposed

to extensive data from heterogeneous sources, learning to recognize linguistic patterns and semantic relations without direct supervision.

Subsequently, fine-tuning allows the model to be adapted to specific tasks, such as text-to-SQL, code generation, or answering questions in specific contexts. This process can be performed with annotated datasets or by exploiting few-shot learning strategies, in which the model learns from a limited number of examples.

A recent approach gaining popularity is Retrieval-Augmented Generation (RAG) [17], which combines the retrieval of relevant information from external sources with the generation of text and thus inserts this information into the input prompt. This method improves the accuracy of responses, reducing the risk of hallucinations and increasing the model's ability to provide up-to-date information.

Scale-up fashion

Scaling up LLMs follows a fundamental principle: increasing the number of parameters, the amount of data and the computational power generally leads to better performance. This hypothesis has been confirmed in several studies, including OpenAI's famous paper "Scaling Laws for Neural Language Models" [18], which showed that larger models tend to generalize better and solve complex tasks more accurately.

In recent years, the number of parameters in LLMs has grown exponentially, from models with a few billion parameters to models with over 500 billion. In parallel, the expansion of training datasets has enabled these models to become increasingly capable of understanding human language fluently and coherently. However, this growth also brings significant challenges in terms of computational costs, efficiency, and energy sustainability.

Capabilities

Large Language Models have a set of emerging capabilities that make them outstanding tools for a wide range of applications. A fascinating aspect is their ability to adapt to context and improve their responses through subsequent interactions, making them particularly effective for advanced conversational applications.

In recent years, some LLMs have demonstrated inductive and deductive reasoning skills, solving logical problems previously considered out of their reach. Unlike traditional models, which required specific datasets and targeted training, LLMs can be used in different contexts without the need for

explicit fine-tuning. This is made possible by the use of techniques such as few-shot learning or in-context learning. Nevertheless, these models may encounter difficulties in correctly interpreting questions that are ambiguous or outside their domain of knowledge, revealing limitations in the deep understanding of concepts.

An important recent development is the introduction of multi-modality, which enables LLMs to process not only text but also images, audio, and video. Models such as GPT-4o, Gemini, and Claude are demonstrating the ability to combine different input modalities to offer richer and more contextualized responses. This evolution opens up new possibilities, such as the generation of images from text descriptions, enhanced speech synthesis, and advanced understanding of visual content. In the business environment, multimodal models could revolutionize the field of customer support and intelligent user interfaces, offering smoother and more intuitive interactions with users.

Limitations

Despite their advanced capabilities, LLMs have several limitations. One of the main problems concerns their knowledge of the world, which is limited to the data they have been trained with. This means they are not always up-to-date with the latest information and may generate inaccurate or outdated answers.

Another challenge is the phenomenon of hallucinations, in which the model produces incorrect or fictional information with great confidence. This problem is particularly critical in contexts where accuracy is paramount.

Finally, LLMs are often affected by bias inherited from training datasets. Research is exploring different strategies to mitigate these effects, but the problem remains open and represents a major ethical challenge.

SOTA Open-source and Proprietary Models

Currently, the LLM landscape is dominated by a mix of proprietary and open-source models. Among the main proprietary models are GPT-4o [6] by OpenAI [19], Gemini [20] [21] by Google DeepMind [22], and Claude [23] by Anthropic [24], while open-source models include LLaMA by Meta [25] [26] [27], Mistral [28], or the brand-new DeepSeek [29].

Open-source models are gaining popularity due to their accessibility, the possibility of being customized for specific applications, and, above all, the possibility of hosting them on one's own clusters. Meanwhile, proprietary

models continue to offer the best performance thanks to unique datasets and infrastructures. The balance between accessibility and performance will be a central issue for the future of LLMs in both academic and enterprise environments.

2.2 Databases and challenges

Databases provide the fundamental infrastructure for the structured storage and management of information, offering an efficient way to store, query, and manipulate data. Unlike NLP systems, which operate in a flexible and highly ambiguous context, databases are designed to handle data rigorously and deterministically, using predefined schemas and formal query languages.

The main categories of databases include relational (SQL) and non-relational (NoSQL) databases. Relational databases, such as PostgreSQL (`https://www.postgresql.org`), MySQL (`https://www.mysql.com`), and Microsoft SQL Server (`https://www.microsoft.com/en-us/sql-server`), organize data into tables with well-defined relationships and use SQL (Structured Query Language) for query operations. This structure allows great consistency and data integrity but can be complex for non-technical users, especially when queries require joins between multiple tables or advanced conditions.

On the other hand, NoSQL databases, such as MongoDB (`https://www.mongodb`), Cassandra (`https://cassandra.apache.org/_/index.html`), and Redis (`https://redis.io`), have been developed to handle unstructured or semi-structured data, offering greater scalability and flexibility in contexts where the rigidity of SQL schema can be a limitation. These databases are well suited to scenarios such as the analysis of big data and real-time applications, however, they are not as well suited for operations that require strong transactional consistency.

2.2.1 Database Interaction Systems

This thesis will focus on improving the interaction between users and relational databases, which will, therefore, be the main focus of our analysis.

After installing the necessary servers or drivers for hosting the database, data can be accessed directly from the terminal of the machine on which the database is hosted. However, for more intuitive use, it is common to use

SQL interfaces, business intelligence tools, or web applications with interactive dashboards. Although these tools improve usability, they still require technical knowledge of SQL or the database structure, thus representing an obstacle for less experienced users.

To facilitate the interaction between users and the database, several approaches have been developed, including:

- **Graphical user interfaces (GUIs)**: tools such as pgAdmin (`https://www.pgadmin.org`) and SQLServer Management Studio (`https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms`), specific to certain database providers, or more flexible solutions such as DBeaver (`https://dbeaver.io`), which support multiple connections to different providers, allow users to explore databases and build queries via visual interfaces, reducing the need to write SQL manually.

- **Advanced NLP-based systems**: some advanced platforms such as Oracle Autonomous Database (`https://www.oracle.com/autonomous-database/`), with Select AI, allow users to formulate natural language queries to obtain answers from databases, improving accessibility for users without technical skills. In addition, some existing platforms are now integrating artificial intelligence to make interaction with databases even more intuitive.

These tools have the potential to bridge the gap between users and databases, simplifying access to information and making data more usable even for those without direct experience with SQL. However, several challenges still remain, such as query accuracy, language ambiguity, and the adaptability of these systems to databases with complex structures.

Chapter 3

Related Works

In this chapter, we will focus on the main publications that have contributed to the development and innovations of text-to-SQL, central topic of this thesis. In order to understand the methods by which natural language can be transformed into the rigid structure of SQL, it is essential to carefully analyze these works.

We have already briefly introduced the task in Chapter 1 and discussed the history of natural language processing, as well as the techniques and approaches used in this major branch of artificial intelligence in the previous chapter. We can now focus on the specific task for which we conducted experiments, drawing on and referring to the literature that we will discuss here. In particular, we will examine the datasets, evaluation metrics, and strategies that have significantly impacted recent research and those that still represent the state of the art in this field.

The aim of this chapter is thus to provide an overview of the most relevant breakthroughs in the evolution of this discipline, which are therefore also relevant to the context of our research. Specifically, as datasets, the Spider-Dev benchmark dataset has a significant impact in this thesis. Furthermore, the techniques for improving the automatic generation of SQL queries, such as prompt engineering, Chain-Of-Thought reasoning, in-context and few-shot learning, schema linking and post-generation query optimization approaches, are essential. Finally, as we will discuss later, mention must be made of the DAIL-SQL [7] approach, which was a great inspiration for this work.

3.1 Introduction to Text-to-SQL

Text-to-SQL falls within the NLP task of text generation, or code generation, and aims to automatically convert a query expressed in natural language into a valid SQL query, as its name suggests. In the literature, we can also often find references to this task simply as NL-to-SQL, NL2SQL, or Text2SQL.

The progress in this field of research increasingly allows users to interact with databases without requiring in-depth knowledge of SQL. This approach is particularly useful in business contexts, where non-technical professionals need access to complex data for decision-making processes.

There are several main challenges to this task [30]. First of all, the complexity and especially the ambiguity of human language challenge the whole NLP context. Especially in this context, where a precise sequence of commands has to be produced to obtain results from a database, it is easy for something slightly ambiguous to lead to the generation of incorrect queries.

The other major difficulty concerns understanding the schema of databases. In experimental contexts, these schemas are often well-structured and follow the best practices. However, in real-world contexts, it is possible to deal with complex databases characterized by incorrect constraints, unclear structures, or ambiguous table and column names, making them difficult to interpret even for humans.

A further obstacle is the rigidity and complexity of SQL. Some queries, in fact, may require uncommon constructs that are difficult to handle.

Finally, it is crucial to consider the ability to generalize across different domains. The structure of a database may vary significantly depending on the context of the application. For example, a relational database designed for a given application might store information about users in a table called *Users*, while a business management system might store the same data in a table called *Employees*. Although they refer to the same concept, term and structural differences can pose a significant challenge. Consequently, the ability to adapt to different domains cannot be taken for granted.

3.2 Datasets

Before analyzing the different techniques that have characterized research in recent years, let us look at the various datasets used to evaluate the performance of models and approaches typical of text-to-SQL. We can

subdivide the datasets primarily according to the objective for which they were designed.

The first category is single-domain datasets, which contain data and present queries related to specific domains. Some of these, among many others, are ATIS [31], which includes data on flights; IMDb [32], famous for data on films and cinema; and the recent BULL [33], which contains financial data. This category of datasets has not always been created exclusively for Text2SQL.

Subsequently, there are datasets characterized by multiple domains, with the aim of creating test benchmarks for the models by evaluating their generalization capacity. In chronological order, one of the main ones is WikiSQL [34], a huge dataset with more than 80000 manually annotated NL-SQL query pairs, whose data concerns information of mixed nature on Wikipedia. It was published in 2017 and is one of the largest datasets used in this field. Next, we have Spider (2018) [35], with its subsequent variants Spider-Realistic [36] and Spider-DK [37]. Considering the importance of this benchmark for the thesis project, we will provide more details here.

Spider is characterized by its greater complexity compared to earlier ones. This is due to the presence of advanced queries that include `JOINs`, nested queries, and more articulated commands than, for instance, the simple `SELECT` and `WHERE` of WikiSQL. In fact, it introduces more complex challenges that require the adoption of advanced techniques than previous datasets.

It consists of 200 databases covering 138 different domains. The number of question-query pairs is 10181, all manually annotated, split between the training set (90%) and the dev set (10%). Unlike WikiSQL, which has only one table per database, Spider has a variable number of tables per database, with an average of 5.1. This leads to the additional difficulty of creating queries with `JOINs` and subqueries, making previously used methods ineffective. It should be pointed out that the queries are purposely made in a non-ambiguous manner to avoid poor data collection quality, however, not representing the reality of natural language questions asked possibly by a human.

Finally, we can mention the more recent BIRD (2023) [38], a dataset designed to improve the evaluation of database reasoning skills by NLP models. Unlike Spider and WikiSQL, BIRD is not limited to the accuracy of SQL query execution, but also evaluates aspects such as logical reasoning and question interpretation. Here, the queries are, in fact, more natural and realistic, and therefore, the models have the added challenge of understanding

the context and the association between words and database entities even better.

Another area of interest concerns datasets built to evaluate the capabilities of models in multi-turn interactions, i.e., situations in which the SQL query is iteratively refined through a series of interactions with the user or model. These include, for example, SParc [39], a multi-turn evolution of Spider to evaluate the ability to capture the context of models, or CoSQL [40] and CHASE [41].

Finally, many other datasets are designed to evaluate specific aspects of text-to-SQL approaches beyond simple accuracy, such as the robustness or computational efficiency of generated queries.

3.3 Evaluation Metrics

We find metrics developed in the literature to evaluate different aspects of SQL query generation. In fact, the queries must mainly be correct from an execution point of view on the database, i.e., return correct data, however, it is also interesting to assess their efficiency and robustness against variations in the natural language query. For this purpose, the principal metrics include:

- **Execution Accuracy (EX)** [35]: measures the accuracy of the results obtained by executing the generated query compared to those obtained by executing the ground truth query. It does not consider the syntactic structure of the query but only evaluates the correctness of the output. It is calculated as:

$$EX = \frac{\sum_{i=1}^{N} \mathbb{1}\left(O_i^{\text{gen}} = O_i^{\text{gt}}\right)}{N} \tag{3.1}$$

- **Exact-Match Accuracy (EM)** [35]: evaluates the accuracy with which the generated query matches the ground truth query, comparing structure and syntax without executing the SQL code. Therefore, it does not take into account possible variations that are syntactically different but semantically equivalent. The metric is calculated as:

$$EM = \frac{\sum_{i=1}^{N} \mathbb{1}\left(Q_i^{\text{gen}} = Q_i^{\text{gt}}\right)}{N} \tag{3.2}$$

- **Component-Match Accuracy (CM)** [35]: this metric is more granular than the previous one and evaluates the correctness of the generated

query by analyzing its main SQL components individually, such as `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `ORDER BY`, and `JOIN`. This metric is thus more tolerant to variations in query syntax, awarding a partial score if some components are correct, even if the entire query does not exactly match the ground truth. Below is the formula:

$$CM = \frac{1}{N} \sum_{i=1}^{N} \frac{\sum_{c \in C} \mathbb{1}\left(Q_{i,c}^{\text{gen}} = Q_{i,c}^{\text{gt}}\right)}{|C|} \tag{3.3}$$

- **Valid Efficiency Score (VES)** [38]: measures the efficiency of the generated query in terms of execution time and optimization compared to the ground truth query. This metric is beneficial for evaluating models that not only produce correct SQL but also generate optimized database queries. The formula follows:

$$VES = \sum_{i=1}^{N} \frac{\mathbb{1}\left(O_i^{\text{gen}} = O_i^{\text{gt}}\right) \cdot R\left(O_i^{\text{gen}}, O_i^{\text{gt}}\right)}{N} \tag{3.4}$$

Symbols Explanation:

- N is the total number of queries evaluated.
- Q_i^{gen} and Q_i^{gt} are, respectively, the generated and ground truth query for the i-th example.
- O_i^{gen} and O_i^{gt} represent, respectively, the output of the generated and ground truth query for the i-th example.
- C is the set of main SQL components.
- $Q_{i,c}^{\text{gen}}$ and $Q_{i,c}^{\text{gt}}$ represent the individual SQL components of the generated and ground truth queries, respectively.
- $R(O_i^{\text{gen}}, O_i^{\text{gt}})$ is a function that measures the execution efficiency of the generated query compared to the ground truth.

In particular, in this thesis, we will carry out the evaluation of our experiments and the various models tested using Spider, so it is essential to talk about the test suites developed to improve the evaluation of this dataset in particular, even though they are not a proper metric. This technique, developed by Zhong et al. [42] in 2020, introduces a method for evaluating the semantic accuracy of text-to-SQL models through the use of distilled test suites. Their approach involves the creation of a small database test suite with dummy data that effectively covers the gold query code, allowing for efficient evaluation of the accuracy of the query execution.

3.4 Evolution of Text-to-SQL Approaches

The evolution of the models and methods used for this task follows the same path described in the previous chapter since it is essentially a natural language processing problem aimed at generating SQL queries.

Initially, the approaches used were mainly rule-based, i.e., based on predefined rules for translating natural language into SQL. An example of this approach is the direct association of terms such as "select", "take", or "extract" with the SQL `SELECT` command. However, such methods are only effective in restricted contexts and have poor scalability, as they require the manual definition of specific rules for each use case.

Subsequently, there has been a transition towards statistical models, which learn the relationships between natural language queries and SQL queries through the use of annotated datasets. In contrast to rule-based approaches, these models do not rely on explicit rules, but instead autonomously learn the correspondences between input and output through the estimation of probabilities from the training data. Although they represent an improvement over rule-based methods, these models continue to exhibit limitations in generalization across different domains and in handling complex databases.

The introduction of neural networks marked a significant advance in this area of research. In particular, RNNs and LSTMs enabled a better handling of ambiguities and complexity in natural language, allowing long-term dependencies between words within complex sentences to be captured more effectively.

Currently, research focuses mainly on the use of LLMs, based on sequence-to-sequence architectures and, in particular, Transformers. Due to the Attention mechanism, such models are able to more accurately identify the relevant parts of a sentence, thus improving the understanding of the natural language query and the association between words and related database entities. In this context, the association between text query terms and SQL components can be interpreted as a link between a query keyword and the name of a table, column, or specific SQL command.

3.5 Advanced Techniques

This section explores the most advanced techniques in the Text-to-SQL field especially concerning the application of deep neural networks and LLMs. The various approaches are not mutually exclusive; rather, they are often

combined to achieve more accurate results.

3.5.1 Prompt engineering

Prompt engineering involves composing input text in order to guide the model to produce the desired output. In the context of text-to-SQL tasks, this means designing prompts that encourage the LLM to generate accurate SQL queries that correctly answer the given natural language questions. By carefully constructing prompts, researchers and practitioners may:

- Provide relevant context about the database schema;

- Include examples of correct SQL translations;

- Specify the desired output format;

- Incorporate domain-specific knowledge or constraints.

The fascinating aspect of prompt engineering is the possibility to obtain excellent results even without performing additional fine-tuning to the model. Thus, this approach has very low costs both in terms of costs both in terms of computational effort and time for inference. Therefore, nowadays it is a widely used technique, useful in business contexts, an area in which this thesis originates, where there is a need for quick and efficient developments.

It is worth noting that prompt engineering's effectiveness can still vary according to several factors, including the specific LLM being used, the complexity of the database schema, the nature and ambiguity of the natural language questions being asked, and the domain of the database.

Having established the importance of prompt engineering in the context of LLMs for text-to-SQL tasks and its challenges, we will now delve into the approaches in the literature applied to build information-rich and contextually relevant prompts.

In-Context and Few-Shot Learning

These techniques, discussed in [43] and [5], permit more accurate generation of SQL queries without the need for specific training, but rather by exploiting contextual examples and additional knowledge to improve the understanding of the natural language query. The model can thus learn precisely from the additional context and N-examples provided.

DIN-SQL [44], for example, uses few-shot learning at different stages by inserting additional information regarding query type and schema linking, or likewise, CodeS [45] inserts metadata for a better understanding of ambiguities.

Some works, such as DAIL-SQL [7] and PET-SQL [46], exploit these techniques by inserting examples of question-query pairs extracted through an example selection mechanism. Precisely, DAIL-SQL masks the names of tables and columns in the question and query before calculating their similarity.

Chain-of-Thought

Chain-of-Thought (CoT) is a technique that aims to improve reasoning in LLMs by incorporating a sequence of logical steps into the prompt to guide the model toward the desired output. This methodology is particularly useful in tasks that require structured reasoning, such as exactly SQL query generation in which the model must understand the user's question, identify entities and relationships in the database, and generate a syntactically and semantically correct SQL query.

The idea behind CoT is to enrich the prompt and, exploiting this technique along with those seen previously and those we will present later, it allows for excellent results. This approach was presented in the work of Wei et al. [47] and later remarked in other works, including that of Zhou et al. [48].

Several studies have explored the application of CoT in the Text-to-SQL domain, taking different approaches depending on the level of reasoning required. Among these, we have basic CoT prompting, which consists of asking the model to break the problem down into sub-steps as in DIN-SQL, or CoT prompting combined with a self-consistency mechanism used, for instance, in C3-SQL [49]. CHESS [50] also uses a series of steps to guide the model in choosing the relevant tables, selecting the database schema, and finally generating the query with possible validations.

Some methods also take advantage of CoT by first implementing a task decomposition step into simpler tasks. For example, in the case of nested queries, methods such as DIN-SQL and MAC-SQL [51] implement a component that can decompose the problem into more straightforward parts that are easier to solve. The former also categorizes queries according to complexity.

3.5.2 Schema Linking

Schema linking refers to the process of mapping words in a natural language question to entities in a database (tables, columns, or even values). Proper schema linking is crucial to avoid ambiguity and generate more accurate SQL. The methods used to accomplish this process are based on different techniques, moving from rule-based to transformer-based methods, thus following the evolution of the whole NLP context.

In particular, we can mention a method proposed in IRNet [52], in which an N-gram algorithm is used to find words within the question that exactly or partially match the names of database tables and columns. This is a simple mechanism that does not consider synonyms and fails very easily in the case of compound names. ValueNet [53] later improves the understanding of some associations by capturing words even in cases of misspellings, but it still does not solve the problems that IRNet presents.

Moving on to approaches based on transformer architectures, we can discuss works such as RAT-SQL [54] and RESDSQL [55]. One of the major contributions of RAT-SQL was using a graph of relationships to represent the structure of the database, in which each node represents a column or table, and the arcs define the connections between them. This paper, moreover, introduces a relational-aware self-attention mechanism, which allows the model to keep track of semantic and structural relationships between query words and database elements. RESDSQL then takes a step further by streamlining the schema linking process and separating it from SQL query generation. It uses a retrieval mechanism to identify similar examples in the dataset and generates a skeleton for the query to be generated later.

Subsequent work explored commercially available modern LLMs, such as OpenAI's GPT family, pointing to the resolution of schema linking through efficient prompt designs that guide the model in identifying tables and columns most relevant to the NL query being asked. DIN-SQL, in this context, guides the model by also incorporating chain-of-thought techniques.

3.5.3 Post-processing optimization

The approaches and patterns described above are often combined, and no less frequently, these are complemented by mechanisms to optimize the SQL query once it has been generated. For example, we have already mentioned self-consistency [56]. This technique has a very simple idea behind it: it requires the model to generate N queries and then evaluate the best candidate.

This evaluation can be done in different ways, such as by majority vote or scoring mechanisms.

Another technique, on the other hand, is self-correction [57], which DIN-SQL and SelECT-SQL [58] use. Unlike the previous one, which sometimes requires a large number of query generations, this technique aims to iteratively improve the generated query through a module or feedback loop that can actually "judge" the correctness of the query. However, this technique is also often wasteful because of the number of iterations. For this reason, in both cases, thresholds are often chosen in order to obtain an excellent trade-off between potential accuracy and computational load.

3.6 State-of-the-art Papers on Spider-Dev

Since the experiments conducted in this thesis are based on the Spider dataset, it is critical to analyze the SOTA approaches developed for this benchmark. The official benchmark ranking is the one reported at `https://yale-lily.github.io/spider`, which collects the best models evaluated on Spider, although submissions were closed in May 2024, as reported on the website. Lately, also, Spider 2.0 [59] has been released for advanced code generation and evaluation of model capabilities even on cloud database systems such as BigQuery (`https://cloud.google.com/bigquery`) or Snowflake (`https://www.snowflake.com`).

At the time this thesis project was conducted, the highest scoring model was MiniSeek, a proprietary system whose technical characteristics were not disclosed. Since no details are available on the methodology used, while acknowledging its ranking in the leaderboard, it is not possible to include it in the analysis.

After MiniSeek, the best publicly available model at the time was DAIL-SQL (Alibaba Group) [7], which introduced innovative few-shot learning strategies with an advanced example selection framework, exploiting a pre-skeleton of the SQL query and the use of special tokens to mask table and column names. This method achieves an execution accuracy of 86.6% on Spider with GPT-4 (fine-tuned) + Self-Consistency.

It should also be pointed out that DAIL-SQL had a significant impact on our work, representing a starting point and influencing part of the adopted methodology.

Subsequently, two new models have achieved particularly relevant results:

- **XiYan-SQL** [60]: This model, also from Alibaba Group, has recently

achieved state-of-the-art results not only on Spider (EX: 89.65%) but also on the BIRD dataset (EX: 75.63%), demonstrating high generalization capability across different text-to-SQL benchmarks. XiYan-SQL introduces a multi-generator ensemble to generate different queries and select the best one, combines in-context learning with supervised fine-tuning, and exploits new techniques for schema linking. In addition, the team of researchers is actively contributing to the scientific community, releasing various open-source models and fostering the progress of the entire text-to-SQL field, as seen from their GitHub account.

- **PET-SQL** [46]: Another recent model that achieved excellent results on Spider (EX: 87.6%), leveraging a different prompt representation, the generation of a preliminary SQL, and the refinement of the final query via cross-consistency between different LLMs.

The continuous development of models such as XiYan-SQL, PET-SQL, DAIL-SQL, and other state-of-the-art systems demonstrates how rapidly the field is evolving, always offering new opportunities for improvement.

Chapter 4

Experiments and Model Evaluation

This chapter analyzes the experiments that contributed significantly to this thesis. As highlighted in the previous chapter, recent advances in the field of Text-to-SQL rely on a wide range of techniques aimed at improving the accuracy of query generation. In particular, Large Language Models are strongly influenced by the prompt provided as input, thus making the construction of effective and informative prompts crucial. However, this does not imply that all available information must be included. Therefore, a strategic balance is required in its selection and organization. Our experimental contribution falls exactly in this context, focusing on prompt engineering. In particular, we take as a starting point the work presented in the paper on the DAIL-SQL method, discussed in the previous chapter, with the aim of extending it in several directions. Firstly, we tested newer models, not considered in the original study, better suited to the requirements of our business use case, which will be discussed in more detail in the following chapter. Next, we experimented with the enrichment of some of the prompts proposed in the reference paper. Finally, after selecting a specific model, we designed and evaluated prompts based on chain-of-thought reasoning techniques to analyze their impact on the quality of query generation.

4.1 Dataset and Evaluation Metrics

To evaluate the performance of our experiments, we used Spider, a dataset widely adopted in text-to-SQL research. Spider is a complex, cross-domain

benchmark designed to test the ability of models to generalize in generating SQL queries from natural language queries. One of its distinguishing features is that the databases in the test set do not appear in the training set, forcing the models to generalize instead of merely storing patterns or simply table and column names already seen in the training phase.

We followed the standard split of the dataset between the training set and the test set (or dev set). The former was used to extract similar questions and queries to be included in the prompts for few-shot learning. In contrast, the latter was used for the final evaluation and comparison with baselines present in the literature. To measure the quality of the generated queries, we adopted the standard metrics in the field of Text-to-SQL, namely:

- Exact Match (EM): measures the percentage of generated queries that exactly match the ground truth query. It is a rigorous metric, as it does not take into account semantically equivalent alternative queries.

- Execution Accuracy (EX): checks whether the generated query produces the same result as the ground-truth query when executed on the database. Unlike EM, this metric is more flexible, as it accepts queries written in different ways as long as they return the same output.

We focus mainly on Execution Accuracy and Exact Match, as these are the metrics most indicative of the model's actual ability to produce correct and functional queries in an application context. To improve the reliability of the evaluation, we adopted the Test Suite Accuracy method. Based on the automatic generation of dummy databases on which to test the queries, this approach reduces the percentage of false negatives and guarantees a more accurate measurement of the model's performance.

4.2 Models Overview

For our experiments, we selected four proprietary models based on key business requirements such as inference speed, cost per token, and provider reliability. The choice reflects the goal of evaluating models that could potentially be used in future in-house applications, including the platform that will be presented in the following chapters. Specifically, we tested models from OpenAI and Google, respectively, GPT 3.5 Turbo and GPT 4o Mini for the first and Gemini 1.5 Flash and Gemini 1.5 Pro for the second. Among these, GPT-3.5 Turbo was included to replicate the results obtained on the

DAIL-SQL benchmark, thus ensuring an internal baseline comparable with previous studies. A summary description of the models now follows, which is useful for comparison during the discussion and analysis of the results.

Model Name	**Knowledge Cutoff**	**Parameters**	**Context Window**
GPT 3.5 Turbo (0125)	Sep 2021	∼175B	∼16k
GPT 4o Mini (2024-07-18)	Oct 2023	∼8B*	128k
Gemini 1.5 Flash	May 2024	>8B*	∼1M
Gemini 1.5 Pro	May 2024	∼200B*	∼2M

Table 4.1: Comparison of AI Models (Part 1: General Information)

Model Name	**Cost Input (1M Tokens)**	**Cost Output (1M Tokens)**	**Throughput (tokens/s)**	**MMLU Performance**
GPT 3.5 Turbo (0125)	0.5$	1.5$	-	70%
GPT 4o Mini (2024-07-18)	0.15$	0.6$	97*	82%
Gemini 1.5 Flash	0.075$	0.3$	166*	78.9%
Gemini 1.5 Pro	1.25$	5$	-	85.9%

Table 4.2: Comparison of AI Models (Part 2: Cost and Performance)

* The exact value has not been disclosed; it is an estimate made by other researchers. [61] [62]

As shown in the table, the choice was guided mainly by three factors: cost, performance, and speed. Two of the selected models offer an exceptionally competitive cost/performance ratio compared to other solutions on the market. Despite having a higher cost, one of the models stands out due to its superior performance in benchmarks such as MMLU. Depending on the final application, the choice of the optimal model will have to balance these aspects.

It should be noted that some of the information reported is derived from third-party estimates and analyses, as providers do not always provide full technical details. For this reason, it was essential to supplement the official

data with independent evaluations and internal tests in order to obtain a more reliable picture of actual performance. In our application domain, the cost of input tokens has a more significant impact than that of output tokens. Each query generated requires a prompt of varying length, influenced by factors such as the prompt engineering techniques adopted, the information included in the prompt, and the database size, since the approaches used include the entire database schema in the prompt.

A limit in the context window, therefore, may exclude essential information. However, in single-turn cases, where the query is generated in a single pass, the prompts never exceeded 16k tokens, making the four models equivalent. In contrast, the difference emerges in multi-turn contexts, where it may be necessary to maintain a history of the messages passed to the model.

Another factor to consider is the knowledge cutoff date, i.e., the date up to which the model has been updated. However, in our specific case, this has little impact, as the SQL domain is relatively stable, with variations occurring only in the event of updates in SQL dialects, which are infrequent.

In conclusion, the models' analysis shows various options suitable for different business scenarios. Models such as Gemini 1.5 Flash are particularly suitable for applications requiring high throughput at a low cost, while GPT 4o Mini offers a good balance between performance and the width of the context supported. On the other hand, models such as the Gemini 1.5 Pro are the ideal choice for complex tasks or applications requiring the processing of extensive contexts at higher operating costs. Therefore, the final choice of model will depend on key factors such as the speed of response required by the application, the operating costs and budget, the breadth of context required, and the overall performance.

4.3 Experimental Setup

A significant contribution to our work was provided by the DAIL-SQL paper by Gao, Hang et al. [7]. In particular, the GitHub repository published by the authors proved to be an excellent starting point, allowing us to reuse part of the code within the terms of the license recognizing their work. On this foundation, we have extended and adapted their work for the platform, which will be described later. Before discussing the details of our implementation, let's define the reference scenario on which our tests are based.

The paper analyzes and compares different prompt representation strategies, i.e., different ways of representing the database and its query. For our

experiments, we adopted a single method, the most promising one: Code Representation (SQL). This representation includes in the prompt an SQL statement for each table of the database, showing the entire DDL of the table, including the name of the table itself, the columns with their data types, and all the constraints, particularly primary keys and foreign keys.

Another fundamental component of the prompt concerns examples of questions and queries similar to the one to be analyzed. This aspect falls under the technique of in-context learning, for which it is essential to consider both the method of selecting examples and how they are organized within the prompt. The examples are extracted from Spider's training set, following two methodologies in which the names of tables and columns are masked before calculating the similarity between the question in the training set and the one to be evaluated in the test set. As regards the organization of the examples in the prompt, we follow the approach proposed in the paper, referred as "QA" example representation, whereby the questions and their queries are presented in the following format:

```
/* Some SQL examples are provided based on similar problems: */
/* Answer the following: <QUESTION> */
SELECT <QUERY>

...

/* Answer the following: <QUESTION> */
SELECT <QUERY>
```

The complete starting prompt is, therefore, as follows:

```
[Selected examples in case of k-shot scenarios, k > 0]
/* Some SQL examples are provided based on similar problems: */
/* Answer the following: <QUESTION> */
<QUERY>
...

[Schema Representation]
/* Given the following database schema: */
CREATE TABLE <TABLE-NAME> (
    <COLUMN-NAME> <COLUMN-TYPE> <OPTIONS>,
    ...
    <CONSTRAINTS>
);
```

```
...

[Actual Question]
/* Answer the following: <QUESTION> */
SELECT
```

4.3.1 Few-shot Learning with Masked Questions and Queries

In these experiments, we will evaluate the performance of all four models using two different example selection methods and varying the number of examples included in the prompt. We will start with the basic zero-shot case and then analyze the results in the 5-shot and 9-shot cases.

Before presenting the results, let us first describe the two example selection methods adopted. Both are based on Euclidean distance as a metric to measure the similarity between the question submitted to the model and those in the dataset. Furthermore, in both methods, references to table names, columns, and values are masked within the questions. However, there are some key differences between the two approaches whose names are adopted in the DAIL-SQL paper and this thesis.

- `EUCDISQUESTIONMASK`

  After masking the fields and calculating the most similar natural language questions, k examples are selected and included in the prompt using the QA format described above.

- `EUCDISMASKPRESKLSIMTHR`

  Again, the masking process is identical, but the criterion for selecting examples changes. In this method, a preliminary model is initially used to generate an initial SQL query prediction. The skeleton is then extracted from the generated query, thus excluding table names, columns, and values to calculate a structural similarity with the queries in the training set. Eventually, questions are selected based on their semantic similarity and similar query structures, ensuring a more cohesive and impactful understanding. A threshold is nevertheless considered for the similarity of the query skeleton.

The paper shows that the second method tends to produce better outcomes than the first. However, it is evident that the use of a preliminary model

influences the generation of the final query. Therefore, one of the purposes of these experiments is to assess whether the models considered, which were not tested in the original benchmark, are able to achieve good accuracy without relying on this auxiliary model. To test the second variant based on query similarity, we will directly use the results obtained from the Graphix preliminary model, as reported in the paper.

4.3.2 Enhance Context with `INSERT` Statements

In this set of experiments, we enriched the prompt by adding `INSERT` statements to provide the model with more detailed context on the actual contents of the database. The idea behind this strategy is to integrate a few rows of actual data in addition to the DDL of the tables and the examples of queries. This data, represented via `INSERT INTO` statements, gives the model a more complete picture, potentially improving the understanding and generation of SQL queries.

The inclusion of real data in the prompt aims to test whether the model can benefit from additional contextual information. In particular, we expect that the presence of meaningful rows of data will help it to better understand the relationships between tables, interpret the semantic meaning of queries, and, most importantly, recognize the specific format of certain fields, such as dates or enumerated values (ENUM). This enrichment could lead to a greater ability to generalize the model, improving the queries' quality. The final format of the prompt, in this configuration, thus includes the table structure (DDL), a set of example questions and queries, and a limited number of rows of actual data, as shown below.

```
[Selected examples in case of k-shot scenarios, k > 0]
...
[Schema Representation]
...
[Samples of Actual Data]
/* Here you have some insert examples: */
INSERT INTO <TABLE-NAME> (<COLUMN-NAME>, ...) VALUES (<VALUE>, ...);
...
[Actual Question]
...
```

For the selection of sample data, we adopted a simple criterion: randomly extract a small number of significant rows for each table. We tested two

configurations, including either 1 or 3 rows of data per table. This approach provides valuable context without making prompt and data processing excessively long and complex.

A further extension of this work could concern the non-random data selection to be included in the prompt. In particular, a possible future development would be the adoption of strategies based on a statistical analysis of the database, preferring, for example, the most frequent data or particularly representative aggregates. This approach would provide the model with an even more informative context, highlighting recurring patterns in the data.

Finally, to assess the impact of this technique, we compared the models' performance by testing two example selection strategies. The analysis was conducted in the zero-shot (as baseline) and 5-shot scenarios to better understand the influence of context enrichment on the model's capabilities.

4.3.3 Chain-of-Thought Prompting

In this section, we analyze the effectiveness of Chain-of-Thought (CoT) prompting techniques applied to our experiments using Gemini-1.5-Flash with a 5-shot approach, 1 `INSERT` example, and the `EUCDISMASKPRESKLSIMTHR` selector. In the discussion of the results, we explain the reasons that led us to these choices. Overall, however, we can state that, as these experiments aim to evaluate the prompts themselves, we prefer to use the aforementioned Google model for inference time and test costs.

This strategy aims to improve the generation of SQL queries through a step-by-step reasoning process, subdivided into several logical steps, to achieve greater accuracy and understanding of the relationships between the database tables and data. We have, therefore, defined and tested different types of CoT prompts, each offering a different approach to the reasoning process and generation of SQL queries.

Below, we present a detailed description of each type of prompt, or more generally, reasoning process, evaluated.

1. **`AHEAD_COT`**: This approach, in a single prompt, suggests the model to follow steps in an orderly fashion before generating the final query. Key steps include understanding the query, identifying the relevant tables and columns, formulating the SQL structure, adding conditions and joins, and finalizing the query. This approach focuses on organizing the reasoning in a clear and sequential manner.

2. **AHEAD_COT_WQUESTS**: A variant of AHEAD_COT that includes an additional step in which the question is restated in three simpler versions. The final query must correctly answer all the reformulated questions. These simplified versions allow the model to solve a simpler problem while emulating a kind of self-consistency on the question side rather than on the generated query side.

3. **SQ_PROMPT_V1**: This technique is based on a prompt decomposition approach, where the model reformulates the initial question into simpler versions (typically three). Unlike the previous prompt, however, it requires the model to follow specific rules rather than actual steps. Subsequently, in a second prompt, we provide the LLM with both the database schema and the restated questions to generate an SQL query that satisfies all the questions.

4. **AUTO_REASONING_V1**: In this approach, the model is asked to generate a sequence of logical steps to construct the SQL query. Once these steps are defined, they are used to guide the generation of the final query by making a second call to the LLM separate from the first. The objective of this technique is to ensure that each step is established and then followed by the model itself, thus following a reasoning process not created by us and potentially different for each request.

5. **AUTO_REASONING_HISTORY_V1**: Similar to AUTO_REASONING_V1, this variant, however, preserves the history of the conversation, allowing the model to interact with the previous message in which it generated the sequence of steps to be followed. This approach is helpful in cases where the initial logical steps need to be corrected or adjusted to improve the quality of the final query.

6. **REPROMPT_EXP_CORRECT**: This approach prompts the model multiple times. Specifically, the model is asked to identify and describe the relevant tables for the query, then generate a first attempt at an SQL query, and eventually validate the query using the database schema. If the validation fails, the model is prompted to correct the error and generate a new query. We could follow a validation and regeneration loop; however, the query correction is not done following the execution on the database because this would invalidate the test. This approach could reduce errors in query generation, as it introduces a process, albeit a simple one, of validation and correction.

7. `REPROMPT_EXP_CORRECT_GENNQUESTS`: Variant of `REPROMPT_EXP_CORRECT` which includes rephrased questions in the reasoning process using the same prompt as the `SQ_PROMPT_V1` method. The final query must correctly answer all reformulated questions. This approach thus includes generating further questions as in other previous approaches. It offers an additional level of difficulty, although it may bring benefits.

The prompts and detailed procedures are in the appendix for the sake of readability. Regardless, we can briefly summarise some key common aspects of several proposed CoT prompts.

N Simplified Versions of the Original Question Some techniques, such as `AHEAD_COT_WQUESTS`, `SQ_PROMPT_V1`, and `REPROMPT_EXP_CORRECT_GENNQUESTS`, include the generation of n = 3 simplified versions of the original question. This step could be useful for simplifying the problem by providing the model with three technically equivalent questions, which must, therefore, also obtain the same answer to be satisfied simultaneously. The underlying concept is a sort of augmentation or self-consistency applied to the question rather than the query. However, not all strategies adopt this technique, as it may entail a higher computational cost without necessarily guaranteeing an improvement in query quality.

Self-Validation (without query execution) Two approaches, `REPROMPT_EXP_CORRECT` and `REPROMPT_EXP_CORRECT_GENNQUESTS`, implement a form of query self-validation. In these cases, the model verifies the query's correctness against the database schema before finalizing the response. This mechanism could reduce the probability of logical or syntactic errors without requiring the execution of the query.

Multi-turn prompts Some of these approaches require multiple requests to the model before generating the final query, making them multi-turn prompts. This means the model follows a sequence of interactions to progressively build the SQL query rather than generating it immediately at the first prompt. Only `AHEAD_COT` and `AHEAD_COT_WQUESTS` are single-turn since the query is generated directly in response to the first input. All other methods require at least a second interaction with the model, with significant variations in the number of steps and input tokens required. `SQ_PROMPT_V1`, `AUTO_REASONING_V1`, and `AUTO_REASONING_HISTORY_V1` involve a first phase

in which the model generates intermediate elements (e.g., queries or logical steps), followed by a second call in which the final SQL query is produced. `REPROMPT_EXP_CORRECT` typically uses 3 or 4 rounds, as the model generates an initial query, checks it against the database schema, and corrects it in further iterations if necessary. Finally, `REPROMPT_EXP_CORRECT_GENNQUESTS` extends the process further, going up to 4 or 5 rounds since, in addition to validating and correcting the query, it also incorporates reformulated queries.

Chapter 5

Results and Discussion

This chapter presents and analyzes the results obtained through the experiments described in the previous chapter. The main objective of our analysis is to evaluate the effectiveness of different prompt engineering strategies applied to the generation of SQL queries using advanced language models. Recall that the comparison between models is carried out using Exact Match (EM) and Execution Accuracy (EX) metrics to measure the quality of the generated queries against the reference queries of the Spider dataset.

For each experiment, the leading data collected will be described and analyzed, comparing the performance of the various models in terms of accuracy, robustness, cost, and timing. Furthermore, the main criticalities that emerged and possible margins for improvement will be highlighted. Based on the data obtained, we will then draw some conclusive considerations on the effectiveness of the various strategies adopted and their potential use in real applications in the following chapter.

Let us now proceed to the detailed analysis of the results of each experiment.

5.1 Few-shot Learning with Masked Questions and Queries

In this section, we analyze the results obtained from the models tested with the two example selection strategies, `EUCDISQUESTIONMASK` and `EUCDISMASKPRESKLSIMTHR`, with different few-shot learning configurations (0, 5, and 9 examples). The objective is to evaluate the improvement in SQL query generation as a function of the number of examples in the prompt and

the selection technique adopted.

Proprietary Models	K-Shots	EUCDISQUESTIONMASK		EUCDISMASKPRESKLSIMTHR	
		EM	EX	EM	EX
GPT-3.5-Turbo	0	0.1934	0.6160	0.1934	0.6160
	5	0.5889	0.6885	0.4690	0.6489
	9	0.5870	0.6895	0.6489	0.7040
GPT-4o-mini	0	0.2350	0.6760	0.2350	0.6760
	5	0.5918	0.6837	0.6460	0.7098
	9	0.6054	0.6808	0.6605	0.7050
Gemini-1.5-Pro	0	0.4864	0.7127	0.4864	**0.7127**
	5	0.6179	0.7408	0.6460	**0.7466**
	9	0.6344	0.7427	0.6721	**0.7591**
Gemini-1.5-Flash	0	0.3249	0.6750	0.3249	0.6750
	5	0.5725	0.7147	0.6411	0.7166
	9	0.6063	0.6934	0.5580	0.7379

Table 5.1: Evaluation of proprietary models across K-shot values using `EUCDISQUESTIONMASK` and `EUCDISMASKPRESKLSIMTHR` example selection strategies.

5.1.1 Comparison of Example Selection Strategies

From the results shown in the table, some key trends emerge.

Increasing the number of examples improves performance For all models, both Exact Match and Execution Accuracy tend to improve as K increases, confirming the effectiveness of few-shot learning over the zero-shot case. However, the gain in accuracy between 5-shot and 9-shot is often marginal, suggesting a potential plateau in the usefulness of adding more examples.

Best results with the `EUCDISMASKPRESKLSIMTHR` strategy This technique, which includes a selection of examples based on the structural similarity of the generated queries, systematically outperforms the `EUCDISQUESTIONMASK` variant. The improvement is evident on all models, except in the 5-shots case, where GPT-3.5-Turbo shows an ambiguous decay. This could be due to the selection of different examples from the other variant, which confuses the model. Regardless of the number of shots, a significant

improvement is observed for GPT-4o-Mini, with an average increase of 2.5% in EX and over 5% in EM. The Google models also show a reasonable margin of improvement in execution accuracy, especially in the 9-shot case: the Pro model improves by 1.7%, while the Flash shows an exceptional 4.5% increase.

Model comparison Gemini-1.5-Pro achieves the best overall performance, reaching a maximum value of EM = 67.2% and EX = 75.9% in the 9-shot configuration with `EUCDISMASKPRESKLSIMTHR`. GPT-4o-Mini shows a good balance between the two strategies, with a constant improvement between 0-shot and 9-shot. GPT-3.5-Turbo confirms itself as the model with the lowest performance, especially in EM, which, however, has a lower importance than EX. This suggests a more limited generalization capability than the other models. Gemini-1.5-Flash, on the other hand, is in an intermediate range, with competitive values in EX but a slightly lower EM than the Pro model. In the overall comparison, Google's models outperform OpenAI's models in terms of execution accuracy. OpenAI's models, on the other hand, exhibit almost identical performance between them, with GPT-4o-Mini being a better choice due to higher speed and cost efficiency. Among the Google models, the Pro consistently performs better in EX, outperforming the Flash by about 2-3%, but at a higher cost and inference time.

5.1.2 Token Cost Analysis

The inclusion of examples within the prompt results in a significant increase in the number of input tokens. The table below shows the number of tokens calculated by the two providers as the result of a request to their respective models. It is important to note that each provider uses a different tokenizer, resulting in discrepancies in the token count. The marked difference between the two results was partially unexpected, although the values remain in the tens of thousands.

K-shots	Google	OpenAI
0	$\sim$ 435k	$\sim$ 381k
5	$\sim$ 720k	$\sim$ 648k
9	$\sim$ 945k	$\sim$ 849k

Table 5.2: Token usage by K-shots for Google and OpenAI

The data reported represents an average between two different example selection techniques, whose differences were negligible. The output tokens are not shown in the table, as they do not vary with the number of shots. For Google, the number of input tokens increases linearly, starting at around 435k with 0-shots up to 945k with 9-shots, while the number of output tokens settles at around 36k. For OpenAI, the growth follows a similar trend, going from 381k to 849k input tokens, with around 27k instead. This is crucial from the perspective of production use, as the increase in tokens directly affects query costs. In addition, increased token consumption could impact the model's latency, with possible repercussions in real-time applications.

5.1.3 Key Insights and Takeaways

We can, therefore, conclude by summarizing what emerged from these first experiments in the following points:

- Few-shot learning improves the quality of the generated queries, but with decreasing results after K = 5.
- The `EUCDISMASKPRESKLSIMTHR` strategy is more effective than `EUCDISQUESTIONMASK`.
- Gemini-1.5-Pro is the best performing model, while GPT-3.5-Turbo achieves the worst results.
- Increasing the number of examples in the prompt increases the number of tokens, which has implications for cost and inference time.

These results suggest that a 5-shot setup with selection based on query similarity might be the most effective choice to achieve the best balance between accuracy and cost.

5.2 Enhance Context with `INSERT` Statements

In this section, we analyse the impact of including real data in the prompts in the form of `INSERT` statements, to provide the models with a richer and more detailed context on the structure and content of the databases. We tested this strategy in both the zero-shot (baseline) and 5-shot cases, evaluating the effectiveness of the approach with 0, 1 and 3 rows of data per table.

Proprietary Models	K-Shots	Selector Type	Insert Samples					
			0		1		3	
			EM	EX	EM	EX	EM	EX
GPT-3.5-Turbo	0	N/A	0.1934	0.6160	0.2388	0.6179	0.2475	0.6218
	5	EUCDISQUESTIONMASK	0.5889	0.6885	0.4941	0.6682	0.5183	0.6721
	5	EUCDISMASKPRESKLSIMTHR	0.4690	0.6489	0.5183	0.6702	0.5261	0.6760
GPT-4o-mini	0	N/A	0.2350	0.6760	0.2814	0.6827	0.2649	0.6789
	5	EUCDISQUESTIONMASK	0.5918	0.6837	0.6025	0.6982	0.5967	0.7176
	5	EUCDISMASKPRESKLSIMTHR	0.6460	0.7098	0.6411	**0.7272**	0.6441	**0.7330**
Gemini-1.5-Pro	0	N/A	0.4864	0.7127	0.5203	0.7466	0.5251	0.7359
	5	EUCDISQUESTIONMASK	0.6179	0.7408	0.6489	0.7649	0.6703	0.7643
	5	EUCDISMASKPRESKLSIMTHR	0.6460	0.7466	0.6634	**0.7823**	0.6905	**0.7756**

Table 5.3: Performance metrics across models, K-shots, and insert samples.

5.2.1 Impact of Real Data Insertion in Prompts

Improvements in query generation The addition of `INSERT` statements increases Exact Match and Execution Accuracy for all models tested. This effect is evident in both zero-shot and 5-shot settings, indicating that the improvement is not directly related to the number of question-query examples in the prompt. Comparing these results with those above, it is observed that adding data further improves performance in few-shot learning (K = 5). The most significant increase occurs on Gemini-1.5-Pro, which reaches EM = 69.05% with three rows of data and EX = 78.2% with a single row. The latter value represents an increase of +3.5% compared to the case without data.

Decrease in performance increasing sample data Including a single row of data results in a significant improvement in EX compared to the basic configuration without data. However, the increase between 1 and 3 rows is often less noticeable or, in some cases, even negative. The addition of more data rows seems to favor EM more, albeit with improvements limited to a few percentage points in the best cases. Execution Accuracy, on the other hand, hardly benefits from the increase from 1 to 3 rows, probably because the random data selection from the database does not provide useful information to the model. On the contrary, it may introduce noise into the prompt, unnecessarily expanding the context and reducing the effectiveness of the prompt itself.

Model comparison Also, in this set of experiments, the best configuration is confirmed to be the 5-shot configuration, using the example selection strategy `EUCDISMASKPRESKLSIMTHR`. In addition, Google's models continue to distinguish themselves with the best performance. Gemini-1.5-Pro achieves the highest results in both EM and EX. Compared to previous experiments, the performance gap with the other models widens further. With only one `INSERT` statement in the best configuration, Gemini-1.5-Pro achieves 78.23%, 4.9% behind the second best model, GPT-4o Mini. A fascinating finding is that Gemini-1.5-Pro, in zero-shot mode with a single row of data, matches the previous best performance achieved with five question-query examples. GPT-3.5-Turbo shows improvement, but still lags behind the other models.

5.2.2 Token Cost Analysis

The considerations regarding tokens fully mirror what was discussed in the previous section, as the insertion of `INSERT` statements again significantly impacts the number of input tokens processed. In addition, output tokens are not reported in the table, as their number remains unchanged regardless of the number of shots or rows of data inserted.

	Google			**OpenAI**		
Data samples	0	1	3	0	1	3
K = 0	$\sim$ 435k	$\sim$ 731k	$\sim$ 1.24M	$\sim$ 381k	$\sim$ 631k	$\sim$ 1.06M
K = 5	$\sim$ 720k	$\sim$ 1.03M	$\sim$ 1.54M	$\sim$ 648k	$\sim$ 898k	$\sim$ 1.33M

Table 5.4: Token usage by K-shots and Insert Samples for Google and OpenAI

For Google, the number of tokens increases to $\sim 1.24M$ in zero-shot and $\sim 1.54M$ in 5-shot with the insertion of 3 rows of data. For OpenAI, the behavior is similar.

In both scenarios (zero-shot and 5-shot), the increase in `INSERT` statements leads to a growth in the number of input tokens between 700k and 800k, depending on the provider. However, this increase does not follow a linear trend from 0 to 3 rows, as the data input involves both a fixed part in the prompt (such as the introductory statement or the SQL commands themselves) and a variable part that depends on the content of the data.

Furthermore, since the data is selected randomly, it is not possible to check whether it contains more or less extensive text fields, thus affecting the overall length of the prompt.

5.2.3 Key Insights and Takeaways

Key points that emerged from the analysis of the above results follow.

- Including real data improves performance, with more promising effects in few-shot learning.
- Increasing the number of rows beyond 1 has little impact, suggesting that the model benefits more from a small representative sample rather than a large volume of data.
- The cost in terms of tokens increases significantly, necessitating careful evaluation in real-world contexts, especially for applications with budget constraints or low latency requirements.
- Gemini-1.5-Pro benefits most from additional data, but it is less cost-effective than other models now that input tokens are increasing in volume.

These results suggest that, to balance accuracy and cost, the most effective strategy may be to use a single row of data per table, avoiding token overload without losing significant benefits.

5.3 Chain-of-Thought Prompting

In this section, we investigate the results obtained with different chain-of-thought prompting strategies, tested with the Gemini-1.5-Flash model in a 5-shot configuration, with 1 example `INSERT` and example selection based on `EUCDISMASKPRESKLSIMTHR`. The choice of this configuration is dictated by the results obtained in the previous experiments. Although this model is not the best performing, it represents an excellent compromise between performance and cost. This is in fact the cheapest model of the four tested and, in this series of experiments, we expect a large volume of tokens in both input and output.

These experiments aim to test whether a structured approach to SQL query generation, divided into several logical steps, can improve query quality

over conventional strategies. In this case, the baseline is represented by the results obtained from the model shown in the table above. Specifically, we report EM = 64.5% and EX = 72.43% for the sake of readability.

COT TYPE CODE	TOKENS		RESULTS	
	INPUT	OUTPUT	EM	EX
AHEAD_COT	∼ 1.094M	∼ 44.1k	0.5928	**0.7611**
AHEAD_COT_WQUESTS	∼ 1.138M	∼ 44.7k	0.5696	0.7533
SQ_PROMPT_V1	∼ 1.270M	∼ 249.4k	0.5938	**0.7649**
AUTO_REASONING_V1	∼ 2.017M	∼ 266.5k	0.5328	0.7379
AUTO_REASONING_HISTORY_V1	∼ 2.535M	∼ 264.8k	0.5135	0.7475
REPROMPT_EXP_CORRECT	∼ 3.113M	∼ 238.8k	0.5406	0.7263
REPROMPT_EXP_CORRECT_GENNQUESTS	∼ 4.051M	∼ 314.9k	0.4274	0.6160

Table 5.5: Evaluation of CoT prompting strategies according to token usage and performance results.

5.3.1 Comparison of CoT Prompting Techniques

The observation of the collected data reveals some significant trends.

Better results with more structured and less iterative approaches The SQ_PROMPT_V1 technique performs best with EM = 59.4% and EX = 76.5%, confirming the effectiveness of decomposing the query into simpler versions and generating the query in a second phase. The AHEAD_COT method also shows good results (EM = 59.3%, EX = 76.1%), suggesting that a well-defined sequential guide helps the model to structure the query correctly. They both register an improvement of about 4% in execution accuracy, however they deteriorate in producing a query exactly equal to ground truth. This is an expected result at least in the first case, since the generation of reformulated sentences leads the model to solve a problem that is slightly different from the original one. Finally, we may note that the method AHEAD_COT_WQUESTS, a variant of AHEAD_COT, obtains slightly lower results than the latter.

Self-reasoning methods slightly improve performance `AUTO_REASONING_V1` and `AUTO_REASONING_HISTORY_V1`, implement a logical sequence of steps for constructing the query, and although they do not achieve the best results, they still manage to exceed the baseline. However, it must be remembered that in these settings, the model generates the steps to be followed and therefore has great variability and dependence on them. The slight advantage of `AUTO_REASONING_HISTORY_V1` in EX may indicate that maintaining the reasoning history helps to improve the consistency of the query execution.

Reprompting and multi-turn does not lead to benefits `REPROMPT_EXP_CORRECT` shows limited improvement over the more direct CoT-based strategies, barely managing to equal the baseline. In contrast, the `REPROMPT_EXP_CORRECT_GENNQUESTS` variant, which combines the former with query reformulation, shows the worst performance (EM = 42.7%, EX = 61.6%), suggesting that excessive complexity in prompting may compromise query generation. These results depend on the fact that, before generating the query, the model follows an articulated process by generating an initial response, describing and selecting the relevant tables, then produces a query, validates it and, if necessary, regenerates a new one. However, this sequence of intermediate steps introduces additional complexity, increasing the risk of inaccuracies and cascading errors between one step and the next.

5.3.2 Token Cost Analysis

Using CoT and multi-turn prompts significantly increases the number of tokens used.

The most efficient methods (`AHEAD_COT`, `SQ_PROMPT_V1`) use about 1.1M-1.3M tokens in input, producing 44k and 249k tokens in output, respectively. In this case, the first method is clearly to be preferred.

Techniques based on self-reasoning and reprompting show a significant and even excessive increase in tokens. To give a few examples, `AUTO_REASONING_V1` uses 2M tokens in input, sometimes more than twice as many as the best direct CoTs. Furthermore, `REPROMPT_EXP_CORRECT_GENNQUESTS`, with the highest number of rounds, even reaches 4M tokens in input and 314k in output, with significantly worse performance.

The increased token consumption significantly impacts cost and inference time, making some of these strategies truly impractical in real-world scenarios.

5.3.3 Key Insights and Takeaways

Finally, we can conclude that:

- The most direct and well-structured CoT strategies are the most effective, with `SQ_PROMPT_V1` and `AHEAD_COT` performing best in terms of EM and EX.
- Techniques based on self-reasoning and multi-turn reprompting do not necessarily guarantee improved performance and tend to have a high computational cost for both input and output tokens.
- Using excessive steps or interactions with the model may compromise query quality, especially if the model has to handle too much input information.

These results suggest that, for optimal application, a well-structured prompt with clear guidance is preferable to overly iterative or complex techniques.

5.4 Discussion

The results obtained from the experiments confirmed the central role of prompt engineering in the generation of SQL queries from natural language. The analyses highlighted how different strategies of example selection, context enrichment, and reasoning structuring can significantly impact the quality of the queries produced, with significant variations in terms of Exact Match and Execution Accuracy. However, new challenges and considerations emerge beyond quantitative evaluation metrics as we move from the experimental environment to a real application context.

5.4.1 Considerations of the Results

The analysis of the results revealed some key trends. First, the few-shot learning technique proved effective in improving model performance, with a gradual increase in the quality of queries generated as the number of examples in the prompt increased. However, the marginal benefit tends to decrease after a certain number of examples (typically 5 or 9), suggesting the existence of a saturation point beyond which the addition of more examples does not lead to significant improvements.

The inclusion of real data by means of `INSERT` statements provided a further increase in the quality of generation, especially with regard to understanding the relationships between tables and the format of the data. However, adding too many rows of data did not produce proportional improvements and, indeed, in some cases, had a negative impact, increasing noise and reducing the effectiveness of the model. This result suggests that a balance between the amount of data provided and the model's ability to use it effectively is crucial to obtain the maximum benefit from this technique.

Chain-of-Thought (CoT) prompting strategies showed mixed results. More structured and straightforward techniques, such as `SQ_PROMPT_V1` and `AHEAD_COT`, proved to be particularly effective, improving contextual understanding and the accuracy of SQL generation. In contrast, more complex and iterative approaches, such as those based on self-reasoning and self-validation, have introduced computational overhead and token usage that is often excessive compared to the benefits obtained.

Everything we have discussed has direct implications for the adoption of such techniques in production environments, where the balance between accuracy, cost, and response speed is critical.

5.4.2 Towards a Real-World Application Context

Large Language Models represent extremely powerful tools with great potential, even in real application contexts. Prompt engineering, compared to fine-tuning, constitutes a considerable advantage, as it allows the generalization capability of these models to be easily adapted to our specific use cases. However, it is crucial to consider the divergence between the results obtained under experimental conditions and the difficulties involved in implementing these models in real enterprise environments. The Spider dataset, employed in evaluations, provides clean and well-structured data, whereas, in concrete scenarios, additional challenges arise, including:

- **Noisy and inconsistent data**: In business databases, data may contain errors, duplicates, or missing values that could compromise the quality of query generation.
- **Ambiguities in database structures**: Although best practices suggest precise rules for database design, these are not always observed. The absence of constraints on primary and foreign keys, as well as the use of incorrect formats or data types, may lead LLM to generate incorrect results.

- **Ambiguity in queries**: Real users, especially non-technical ones, tend to formulate queries ambiguously, using synonyms or complex language structures, making it more difficult for the model to interpret them correctly.
- **Presence of numerous different SQL dialects**: Although the experiments focus on the SQL language, it is crucial to consider the existence of numerous SQL dialects depending on the database server provider, such as SQL Server, PostgreSQL, MySQL, and SQLite, among others. This means some commands valid in one system may not be supported in another.
- **Data security and privacy**: In a real-world context, databases no longer contain dummy data, so it is essential to monitor the data flow and processing and ensure that the system remains compliant with data protection regulations.

In this context, it is therefore necessary to develop a more comprehensive infrastructure and implement mechanisms to mitigate these challenges and meet the various requirements.

The main objective of this thesis is to experiment with the latest LLM available on the market and implement a platform, albeit in the form of a Proof-of-Concept, that offers an intuitive interface for querying databases, reducing the complexity of use and improving the user experience.

In this second phase, indeed, we encountered the difficulties previously discussed, which will be analyzed in more detail in the following chapter. The preliminary analysis, in fact, guided the design of the platform, which was developed taking into account the needs of a real application context.

For the choice of the most suitable model, a compromise between accuracy, cost, and inference speed was evaluated:

- **High accuracy**: Gemini 1.5 Pro, with a 9-shot configuration and the use of 1-3 rows of real data, is optimal for scenarios where high accuracy is required, albeit at a greater cost.
- **Efficiency and speed**: Gemini 1.5 Flash, with a 5-shot configuration and a single row of sample data, is instead a good balance of performance and cost.

The second configuration is the one adopted by default in the developed platform.

5.4.3 Final Remarks

The discussion shows that the success of a text-to-SQL system depends not only on the strength of the model used but also on the prompting strategy adopted and the ability to integrate the system into a real enterprise environment. The most effective strategies are based on a balance between structuring the reasoning, selection of examples, and use of actual data while avoiding excessive complexity that could negatively impact the cost and performance of the system.

The transition from the experimental environment to practical implementation requires, therefore, an approach that considers not only quantitative metrics but also qualitative factors such as ease of use, robustness against user error, the ability to adapt to imperfectly structured data, and, last but not least, security, which should be implemented with a by-design and by-default approach.

The developed platform fits in this direction, offering a concrete solution for applying text-to-SQL in a business context. It has the potential to improve and facilitate non-expert users' access to and querying of databases.

In the following chapter, the platform's concrete use case will be illustrated, delving into the technologies adopted, the implementation process, its operational impact, and the possible optimizations that could transform this Proof-of-Concept into a fully integrable and ready-to-use product.

Chapter 6

Use Case and Proposed Solution

6.1 Business Requirements and Challenges

6.1.1 Corporate Landscape And Fields of Application

Access to data is a fundamental need in any business context, regardless of the sector or the user's role. The platform developed responds to this need by offering a centralized system for managing database connections, allowing users to query them intuitively and display the results in multiple formats.

Compared to a simple text-to-SQL conversion via LLM, the platform's objective is to provide an integrated solution that simplifies interaction with the data, minimizing the technical support required and improving operational efficiency.

The project was realized in cooperation with Connect IT Reply, a company specializing in IT consultancy for IoT platforms and connected products. Having already played a role in the company, it was possible to directly identify internal dynamics and seize the opportunity to optimize the operational flow, proposing solutions based on actual needs observed in the daily work environment.

The proposed platform is ideally suited to companies that manage multiple databases that are isolated from each other for security and confidentiality reasons. This is a common situation in the IT consultancy.

In these scenarios, a single point of access to data allows both technical and non-technical staff to operate more quickly and autonomously, reducing the dispersion of resources and activity downtime due to context switching

between different tools.

6.1.2 Categories of users involved

The use of the platform can be beneficial for several professionals, including:

- **Developers**: can speed up the database query process by centralizing access to information and reducing the time spent writing repetitive queries. Often, in fact, a developer engaged in developing a new functionality needs to quickly understand the database's structure to analyze its impact or correctly define its design.
- **Partners, Project Managers, and Business Analysts**: They can extract data without the need for technical support, improving the speed with which they can obtain useful insights for their own activities. In addition, the platform directly returns a graphical response, which is particularly useful in the case of preliminary analyses as it reduces the need for additional tools to visualize the data. Of course, this is not a replacement for software dedicated to the creation of complex dashboards, but it is a valuable tool for an initial exploratory analysis.
- **Corporate customers**: In some instances, they may need up-to-date data without having to contact the technical team for every request. From our point of view, this functionality could become an additional service to be offered to customers, who would use it independently for their internal needs.

6.1.3 Business context challenges

Users with non-technical roles face several obstacles in accessing data:

- **Time and dependence on technical support**: Requesting data from the technical team causes operational delays.
- **Lack of adequate tools**: They often lack software to consult and interpret data quickly.
- **Cost and complexity of existing commercial tools**: Commercial solutions already available on the market often entail high costs or difficulties integrating with internal company policies, making a dedicated internal solution preferable.

The proposed platform addresses these needs, offering a solution that is cost-effective, secure, and easy to integrate into corporate workflows. It also manages the complexity associated with the coexistence of databases with different SQL dialects (e.g., MySQL, PostgreSQL, SQL Server), guaranteeing flexibility and multi-provider compatibility.

6.1.4 Main technical challenges

The implementation of the platform presents several challenges, already discussed in the previous chapter. One of the main difficulties concerns the interpretation of natural language since the queries users ask are often imprecisely or ambiguously formulated, making their correct understanding complex. Another critical aspect is the balance between accuracy and cost, which requires optimization of the generated queries to avoid excessive use of tokens, therefore reducing both costs and response times. Finally, the platform must manage the heterogeneity of business data, characterized by non-standardized patterns, formats, and naming conventions, which complicate its integration and processing.

6.1.5 Proposed Solution Objectives

Given the considerations mentioned above, the main objectives of the platform are: Simplify and speed up data access by removing technical barriers related to the use of SQL.

- Reduce the developers' workload by decreasing interruptions due to data requests.
- Centralize and standardize database access, improving the overall efficiency of the workflow.
- Optimize the presentation of results through intuitive interfaces and versatile visualizations (text, tabular, graphical).
- Maximize the potential of LLM, thanks to targeted prompt engineering strategies for generating reliable queries consistent with user requirements.

The proposed platform thus represents a balanced solution between ease of use, performance, and economic sustainability, designed to adapt effectively to the observed real business context.

6.2 Technological Stack and System Architecture

The platform's development follows a modular and flexible approach. Currently a Proof of Concept (PoC), the architecture is designed to facilitate experimentation and adapt to future evolutions, with the aim of improving scalability, security, and maintainability.

The system's structure allows for a possible extension towards a more robust infrastructure, with the possible use of containers, orchestrators, and advanced authentication mechanisms. This section provides an overview of the main technologies adopted, while the implemented functionalities and the complete flow of requests will be explored in the following chapters.

6.2.1 Backend

The backend is developed in Python and deals with query processing, database integration, and functionalities based on artificial intelligence models. A significant part of the code concerning query preprocessing was borrowed from the work of the DAIL-SQL paper researchers. However, several modifications and adaptations were made for our experiments and their integration with the platform architecture.

For the API deployment, we chose a simple and fast framework, FastAPI, since it is a PoC. APIs are REST, but we envisage the possibility of introducing WebSocket for chat management to make real-time interactions more efficient.

Since the platform currently runs locally, an authentication system has not been implemented. To improve security, JWT and OAuth could be adopted in a future distributed version.

6.2.2 Frontend

The frontend was entirely developed as part of this thesis using React and TypeScript, with Vite, a build tool to optimize project management and speed up project builds. The interface is designed to offer a simple and intuitive user experience without requiring specific technical skills.

Currently, the state of the application is managed locally in the React components, without the use of state management libraries that could be used in the future. The frontend communicates directly with the backend via REST APIs. Chart.js is used for data visualization, allowing a minimalist,

clear and interactive representation of the information.

6.2.3 Database and LLM Integration

The management of new database providers and integration with LLM models were implemented by extending and adapting existing code.

Communication with the database occurs via the `pyodbc` library, which guarantees persistent connections and optimizes the execution of SQL queries. Currently, the platform supports SQLite and SQL Server, with the possibility of extending support to other providers in the future.

For natural language query processing, the backend communicates with the selected model via external APIs using an API key stored in the configuration files. The architecture is designed to be modular, allowing the integration of different models available on the market by simply implementing the methods defined in the dedicated interface. Moreover, the platform natively supports integration with available Hugging Face models, allowing them to be used even in local environments without the need to connect to external cloud services.

6.3 Platform functionalities

The platform offers users a centralized and intuitive environment to access, query, and manage databases through an interface that takes advantage of the advanced capabilities of artificial intelligence models. It consists of four main functionalities that enable it to meet the needs of different types of users while ensuring ease of use and technical flexibility.

6.3.1 Database Connections

The platform offers a centralized system for managing database connections, allowing users to configure and access multiple databases easily and securely. The interface allows users to:

- Add new connections by specifying an identifier, SQL dialect and authentication method, which can be by connection strings or credentials.
- View and manage active connections, providing a clear list of configured databases.

- Remove connections that are no longer needed, with a confirmation mechanism preventing accidental removal.

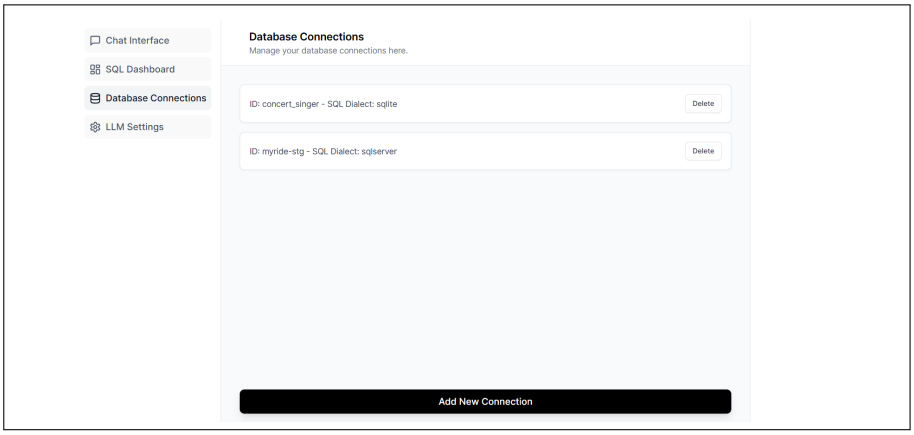

Figure 6.1: Database Connections: List of available connected databases

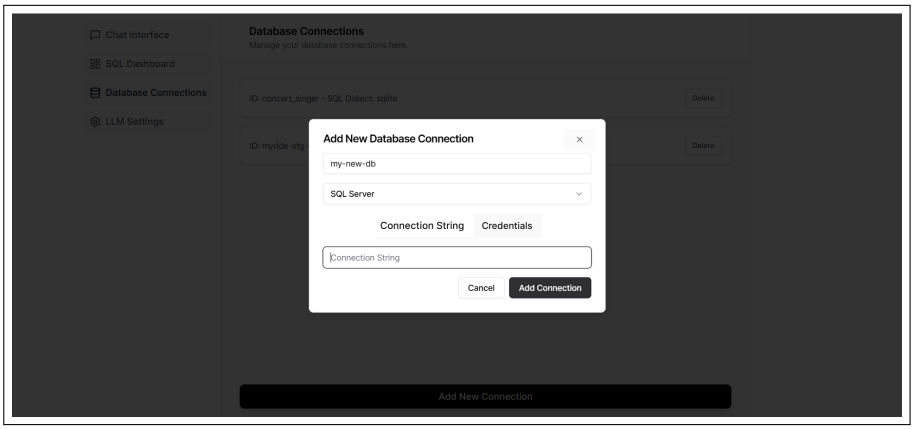

Figure 6.2: Database Connections: Dialog modal view to add a new connection

6.3.2 SQL Dashboard

The SQL Dashboard is the operational heart of the platform, allowing users to formulate natural language queries and execute SQL queries. The interface integrates the advanced capabilities of LLM templates to automatically translate textual queries into SQL queries, providing quick and intuitive access to data. Key features of the SQL Dashboard include:

- **Automatic query generation**: Users can type in a natural language query, select the reference database, and get the corresponding SQL query generated by the LLM model.
- **Query execution and modification**: Query execution is performed directly after the natural language query; however, it can also be manually modified and re-executed, providing flexibility and total control over the queries.
- **Visualization of results in different formats**: Data can be consulted in textual, tabular, or interactive graphical format, facilitating the analysis and understanding of information.
- **Query history**: The platform stores executed queries, allowing users to review, re-execute, or refine previous queries quickly and easily.

The aim of the SQL Dashboard is to combine simplicity and power, offering an interface accessible even to non-technical users without compromising the flexibility required by more experienced users.

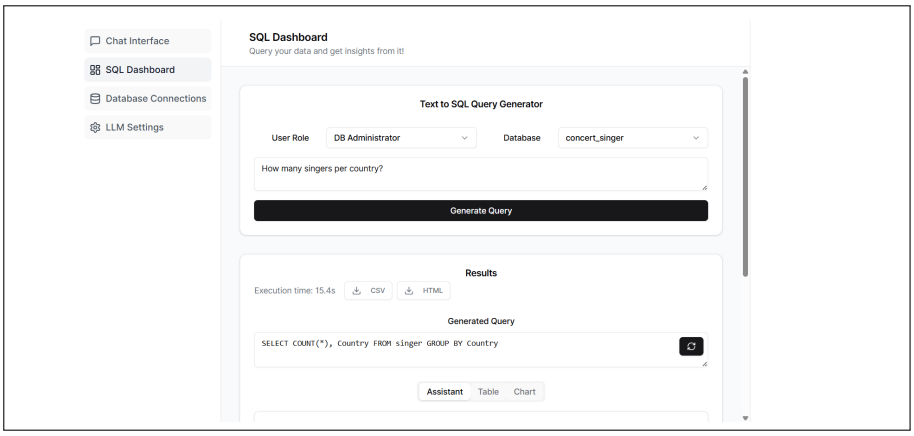

Figure 6.3: SQL Dashboard: Query generation and execution

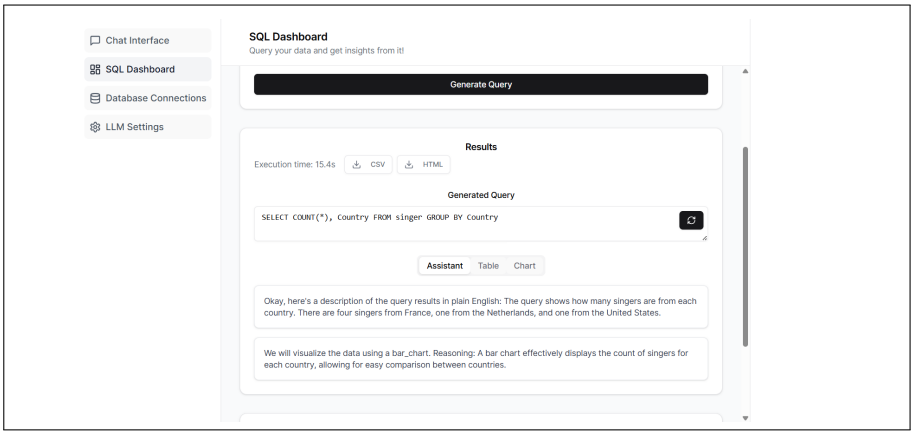

Figure 6.4: SQL Dashboard: Query execution output in natural language format

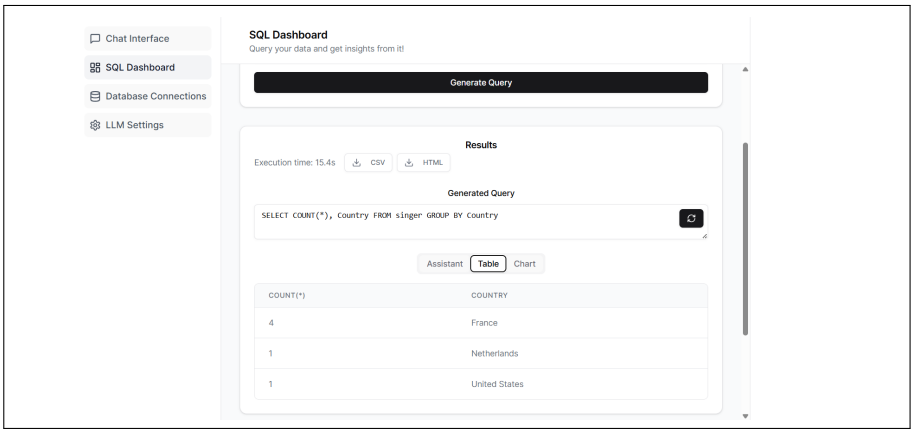

Figure 6.5: SQL Dashboard: Query execution output in tabular format

6.3.3 Chat Interface

The Chat Interface section aims further to facilitate interaction with the database through a conversational interface. The virtual assistant is designed to help users understand the database's structure and suggest possible SQL queries to be executed directly in the Dashboard.

This tool is particularly useful for non-technical users as well as for those who need to quickly familiarize themselves with a database whose internal structure they are unfamiliar with, thus improving their operational

Figure 6.6: SQL Dashboard: Query execution output represented as a chart

autonomy and significantly reducing the time needed to obtain answers and results.

The leading chat functionalities include:

- **Database selection**, to ensure relevant queries to the desired databases.
- **General inquiries for understanding databases**, with markdown text to enhance readability.
- **Generation of SQL queries**, translating textual queries into executable code.
- **Direct execution of queries**, extracted from the conversation, with the ability to execute them and look at the results in the SQL Dashboard section.

6.3.4 LLM Settings

The LLM Settings section allows users to customize the parameters of the AI models used in the platform to generate SQL queries and for conversational features. This configuration enables users to optimize system performance according to specific accuracy, speed, and efficiency requirements. The main configurable options include:

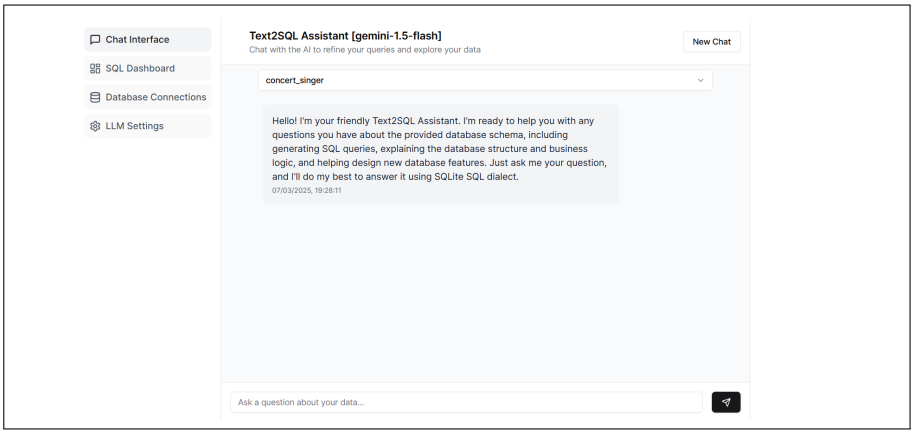

Figure 6.7: Chat interface: Select a database connection and start a new chat

Figure 6.8: Chat interface: Markdown messages, query generation, and "Execute query" button

- **Provider and model**: The user can choose between different providers (e.g., Google, OpenAI) and LLM models (e.g., Gemini 1.5 Flash, GPT-4o-mini).
- **Temperature control**: This parameter influences the creativity of responses, with lower values for more predictable responses and higher values for greater variability.
- **Activation of the Chain-of-Thought technique**: This guides the model through intermediate reasoning, as in the experimental phase.

- **Saving customized settings**: Users can save preferred configurations to apply them in future interactions.

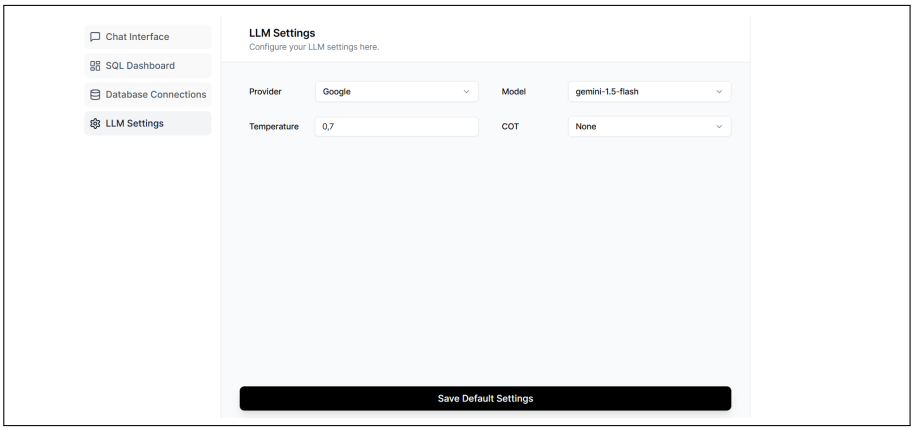

Figure 6.9: LLM Settings: Modify settings according to preferences

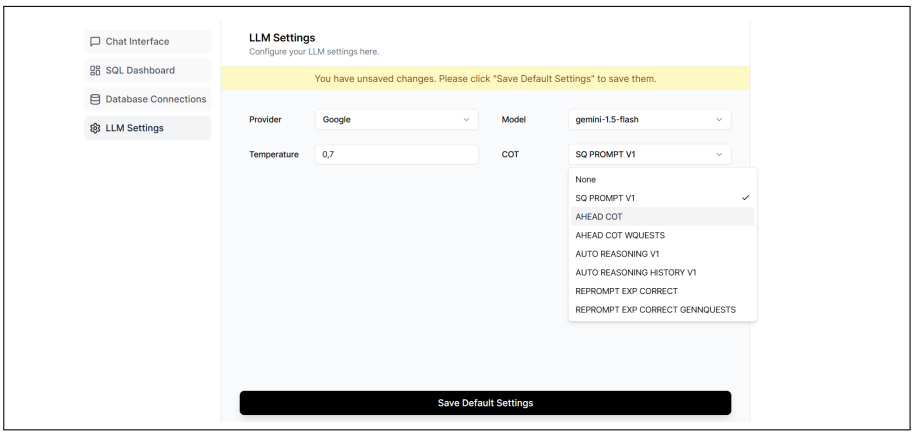

Figure 6.10: LLM Settings: Select among different CoT prompting methods

6.4 Real-time Data Retrieval Process

The SQL Dashboard represents the core of the platform and integrates the various prompting techniques described in previous chapters. The processing of the user's requests follows a well-defined flow, which ensures the correct interpretation of them, the generation of the SQL queries, and finally, the

output of the results for visualization. Figure 6.11 shows a sequence diagram of the whole process, summarizing the following section.

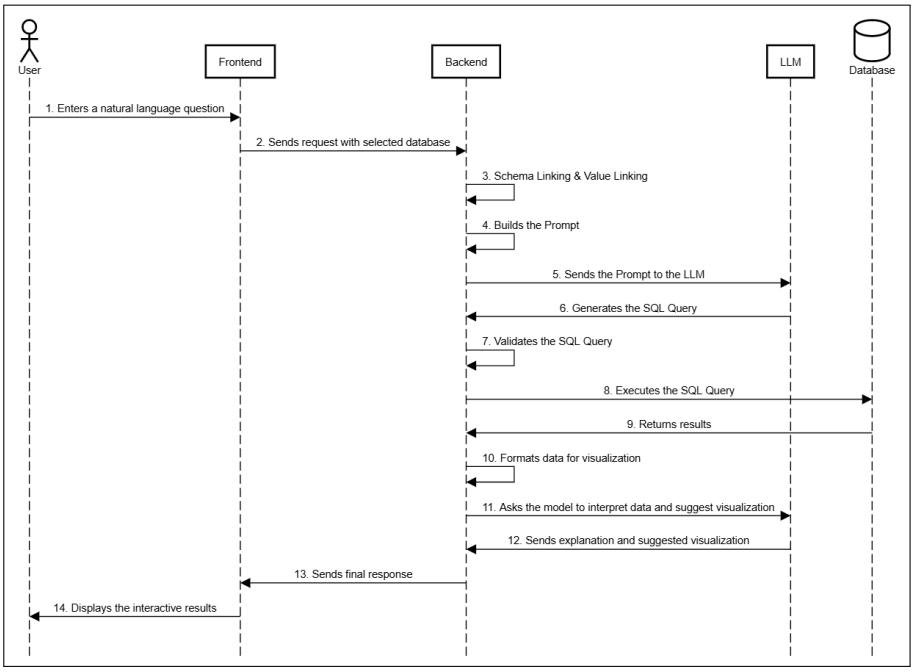

Figure 6.11: Sequence diagram of the user request when using the SQL dashboard

Preliminary Steps Before discussing the details of the request flow, it is essential to describe some preliminary operations that guarantee the platform's proper functioning.

Each time a new database connection is added, the platform analyzes its structure and serializes it in a JSON file, including information on tables, columns, and constraints. In parallel, the system maintains a library of sample databases, which are used to enrich the prompt. These databases and the corresponding queries need to be preprocessed beforehand, calculating the schema linking and serializing data in an accessible format.

Thanks to this strategy, the platform has all the information needed to speed up request processing right from the start, avoiding the need to repeat parsing operations and analysis of the database structure with each query.

Phase 1: Sending the Query and Identifying the Context The interaction begins when the user formulates a natural language query through

the SQL Dashboard interface. The frontend sends a request to the API exposed by the backend, specifying the textual query and the reference database. The platform ensures that the selected database is valid and accessible.

Phase 2: Schema and Value Linking One of the most complex aspects of the conversion from natural language to SQL is schema linking, i.e., the process of associating the keywords in the user's question with database entities, such as tables and columns. In parallel, value linking is performed, allowing numeric values or strings in the question to be identified and mapped to the data contained in the database. For this phase, we adopted the approach described in the IRNet paper. This step is crucial for the example query selection step, as the identified table names, columns, and values are replaced with special masking tokens.

Phase 3: Constructing the Prompt for LLM A structured prompt is built, which includes:

- The database schema representation, with tables, columns, and relationships.
- Sample entries for each table, dynamically extracted from the database.
- The user's original question, keeping the natural phrasing.
- A set of similar queries extracted from the dataset, to better guide the model in generating the query.

Phase 4: Query Generation The prompt is sent to the selected LLM, which generates the SQL query that answers the user's question. The query is then extracted and validated.

Phase 5: Query Validation Before executing the query on the database, the platform verifies that it is read-only, thus avoiding write operations that could compromise the data. In a future evolution of the system, this validation could be adjusted according to user roles, allowing more advanced operations for users with specific authorizations.

Phase 6: Query Execution and Data Retrieval After validation, the query is executed on the database, and the results are retrieved.

Phase 7: Formatting Results for Visualization The system performs further processing to adapt them to the frontend presentation libraries:

- The data are formatted for clear visualization.
- The LLM model is prompted to interpret the data and generate an explanatory textual response.
- The most suitable type of graph is selected to represent the data (only tables, bar charts, line charts, and pie charts), facilitating user analysis.

Phase 8: Saving Query History Each executed query is saved in the history, allowing the user to consult, modify, and re-execute previous queries.

Phase 9: Sending the Response to the Frontend Once the process is complete, the API sends the response to the frontend, which displays the results interactively. The user can explore the data and modify the query, providing a smooth and intuitive experience.

6.5 Further Improvements, Security, and Scalability

This section analyzes some improvements that could make the platform more secure, scalable, and efficient, adapting it to more complex business scenarios.

6.5.1 Query Generation Optimizations

Experiments have shown the need to improve the efficiency and accuracy of the models used to generate SQL queries from natural language. One of the most promising directions concerns the implementation of a Retrieval-Augmented Generation (RAG), allowing the most relevant tables to be selected before the query is generated, reducing the number of tokens consumed, and improving the relevance of the answers.

Another area of optimization concerns database schema caching. Storing the structure of tables and relationships between data would reduce latency in the query generation process, avoiding redundant calls to artificial intelligence models. However,r, a mechanism for automatically updating metadata when the database is updated should be considered.

In addition, fine-tuning models on specific datasets could improve understanding of database structures and expressions commonly used by users, leading to greater accuracy in generated queries.

The platform could then integrate automatic suggestions for query generation, making the experience more intuitive and guided. Furthermore, expanding compatibility with other SQL dialects would extend the scope of the platform, making it more versatile and suitable for different business contexts.

6.5.2 Cost-effective Optimization

The use of AI models for generating SQL queries introduces inference costs that must be balanced against the accuracy of the answers. One possible solution is the integration of a dynamic model selection mechanism, which may choose in real time between lighter, cheaper, or more advanced models depending on the complexity of the request.

Another approach to reduce operational costs could be adopting open-source or on-premise models, eliminating the dependence on external APIs, and providing greater control over computational resources. Finally, the implementation of caching strategies for prompts and responses would avoid repeated calls to the model, improving efficiency, and reducing processing time.

6.5.3 Data security and protection

Strengthening data security is essential in enterprise and production environments. To ensure adequate protection, it would be advisable to introduce encryption systems for credentials and sensitive data, advanced access control mechanisms to manage permissions according to corporate roles, and secure session management through authentication protocols. These improvements make it possible to limit access to sensitive data only to authorized users with controlled access and improved traceability.

6.5.4 Infrastructure Scalability

The current implementation of the platform, being a proof of concept, is designed to be easily extendable at the infrastructure level. To support an increasing number of users and requests, a fundamental first step would be the adoption of scalable cloud infrastructures or serverless architectures,

exploiting technologies such as Docker and Kubernetes, which can efficiently distribute the workload, and cloud providers such as AWS, GCP, or Azure.

A further development could be implementing a microservices structure, further improving scalability and allowing modular management of platform components. This approach would distribute the workload more efficiently, facilitate the integration of new services without impacting the entire system, and optimize the allocation of resources according to operational needs.

Chapter 7

Conclusions

This thesis has addressed the issue of user-database interaction through the use of large language models for the generation of SQL queries from natural language. The primary goal was to develop a platform that simplifies data access for non-expert users, reducing the need for extensive technical knowledge of SQL, and for more technical roles to speed up the querying process and understandability of the database structure.

After a deep analysis of the literature and existing technologies in the Text-to-SQL field, we implemented and tested different prompt engineering strategies, evaluating the performance of different language models with varying configurations. As expected, the experimental results demonstrated that techniques such as Few-Shot Learning, the addition of `INSERT` statements in the prompt, and the use of Chain-of-Thought prompting can significantly improve query generation accuracy.

Among the tested models, Gemini-1.5-Pro performs better, achieving an execution accuracy of 78.2% in the optimal configuration (without CoT). However, its use entails a higher cost compared to other solutions like GPT-4o-Mini or Gemini-1.5-Flash, which offer a better cost-performance ratio even if slightly less performant. From a practical implementation perspective, the developed platform represents an important step forward in facilitating access to enterprise real data, allowing even non-technical users to query complex databases through an intuitive natural language interface.

Some challenges remain open regarding both the model's performance and platform perspectives.

Implementing a retrieval-augmented generation (RAG) system would improve query generation, especially for complex requests, particularly in

databases with a large number of tables. Additionally, common text generation errors, such as model hallucinations, must always be managed, as they can easily lead to incorrect or ineffective queries.

Another issue is also the computational cost and scalability of the system for use in production environments with high request volumes. Furthermore, the integration with heterogeneous databases will further enhance the usability of the platform.

In the future, possible developments include the adoption of open-source models to reduce reliance on proprietary APIs, such as the brand-new DeepSeek [29], the deployment of the platform using cloud provider resources, and the optimization of the user experience.

In conclusion, this work has highlighted the potential of generative artificial intelligence in data management, offering innovative solutions to enhance accessibility and efficiency in database querying for both expert and non-expert users. The results provide a foundation for further research and developments in the Text-to-SQL field, opening up new opportunities for practical applications in both the corporate and academic sectors.

Appendix A

Experimental Setup: CoT Prompts

In this appendix, we describe the prompts and procedures used for each type of Chain-of-Thought involved in generating SQL queries. Throughout the appendix, the term `COMPLETE-PROMPT` refers to the complete prompt presented in Section 4.3. This full prompt, as shown, is divided into key sections, which we will refer to as `QUERY-EXAMPLES`, `DB-SCHEMA`, `INSERT-SAMPLES`, and `QUESTION`.

A.1 AHEAD_COT

In this case, the following prompt is designed for a single interaction with the LLM.

```
"""
    Let's approach this step-by-step:
    1. Understand the question and identify the key elements.
    2. Determine the tables and columns needed.
    3. Formulate the SQL query structure.
    4. Add necessary conditions and joins.
    5. Finalize the SQL query.

    Now, let's go through each step:

    {COMPLETE-PROMPT}
```

```
"""
```

A.2 AHEAD_COT_WQUESTS

This prompt is pretty similar to the previous one plus the addition of a question rephrasing step.

```
"""
    Let's approach this step-by-step:
    1. Understand the question and identify the key elements.
    2. Rephrase the given question in 3 different ways maintaining
   the same meaning but simplifying it.
    3. Determine the tables and columns needed.
    4. Formulate the SQL query structure that satisfies all the 3
   rephrased questions.
    5. Add necessary conditions and joins.
    6. Finalize the SQL query that satisfies all the 3 rephrased
   questions.

    Now, let's go through each step:

    {COMPLETE-PROMPT}
"""
```

A.3 SQ_PROMPT_V1

This is a two-stage approach where, first the LLM generates simplified questions, and then generates the SQL query based on an enhanced prompt.

```
# Step 1: Generate simplified questions
"""
    Rephrase the given question in 3 different ways. Follow these
   rules:
    1. Maintain the original meaning.
    2. Simplify the language so an 8-year-old child could easily
   understand it.
    3. Ensure each rephrasing is distinct from the others.
```

```
7
8       Your response must be a numbered list of the 3 rephrased
        versions without any explanations.
9
10      Given the following database schema:
11      {DB-SCHEMA}
12      {INSERT-SAMPLES}
13      {QUESTION}
14  """
```

```
1   # Step 2: Generate SQL query
2   """
3       {QUERY-EXAMPLES}
4       {DB-SCHEMA}
5       {INSERT-SAMPLES}
6
7       Answer the following:
8       1. {REPHRASED-QUESTION-1}
9       2. {REPHRASED-QUESTION-2}
10      3. {REPHRASED-QUESTION-3}
11  """
```

A.4 AUTO_REASONING_V1

Firstly, the LLM generates reasoning steps as a guideline for the correct SQL query using the following prompt:

```
1   # Step 1: Generate reasoning steps
2   """
3       Given this prompt:
4       {COMPLETE-PROMPT}
5
6       Generate the sequence of actions needed to generate the correct
        SQL query.
7       Your response must be a numbered list of actions.
8   """
```

Then the reasoning steps are injected into another prompt to generate the SQL query.

```
# Step 2: Generate SQL query
"""
    Given the following database schema:
    {DB-SCHEMA}

    Follow this reasoning to generate the correct SQL query:
    {GENERATED-REASONING}
"""
```

A.5 AUTO_REASONING_HISTORY_V1

The initial step is identical to the previous one, where the LLM generates reasoning steps. In the second step, instead, we continue the current conversation to generate the SQL query, with the following simple message:

```
# Step 2: Generate SQL query
"""
    Now write the correct SQL query without explanations.
"""
```

A.6 REPROMPT_EXP_CORRECT

This approach queries the LLM three or four times to generate the correct SQL query. The first prompt is designed to select and describe the relevant tables.

```
# Step 1: Select and describe relevant tables
"""
    {QUERY-EXAMPLES}
    {DB-SCHEMA}
    {INSERT-SAMPLES}
    Select and describe the tables needed to answer the following: {
   QUESTION}
```

```
    Follow this structure to respond:
    1. Relevant tables:
    2. Tables Description:
    3. Relationships among relevant tables if any:
    4. Other useful information:
"""
```

Then a query is generated based on the tables selected in the previous step.

```
# Step 2: Generate SQL query
"""
    {COMPLETE-PROMPT}
    Some useful information you may need to answer:
    {RELEVANT-TABLES-DESCRIPTION}
"""
```

After generating the query, the LLM validates it with the following prompt:

```
# Step 3: Validate the generated query
"""
    Given the following database schema:
    {DB-SCHEMA}
    {INSERT-SAMPLES}
    Question: {QUESTION}
    Query: {GENERATED-QUERY}
    Tell me if the query answers the question correctly. Answer only
    with YES or NO.
"""
```

Finally, if needed, the LLM corrects the query based on the feedback received in the previous step.

```
# Step 4: Generate corrected query
"""
    Then correct the query.
"""
```

A.7 REPROMPT_EXP_CORRECT_GENNQUESTS

Actually, this CoT reasoning is a combination of the previous one and the SQ_PROMPT_V1. In this case the number of interactions can be up to five. The initial step is, in fact, the same as the first step in SQ_PROMPT_V1, described in A.3, where the LLM generates simplified questions. Then, the rest of the interactions are very similar to the REPROMPT_EXP_CORRECT approach, with the addition of the simplified questions in the prompts. Here we show only the prompts which differ from the previous approach.

```
# Step 2: Select and describe relevant tables
"""
    {QUERY-EXAMPLES}
    {DB-SCHEMA}
    {INSERT-SAMPLES}
    Select and describe the tables needed to answer all the
   following questions with a single query: {REPHRASED-QUESTIONS}
    Follow this structure to respond:
    1. Relevant tables:
    2. Tables Description:
    3. Relationships among relevant tables if any:
    4. Other useful information:
"""
```

```
# Step 3: Generate SQL query
"""
    {QUERY-EXAMPLES}
    {DB-SCHEMA}
    {INSERT-SAMPLES}
    Write a single SQL query to answer the following equivalent
   questions: {REPHRASED-QUESTIONS}
    Some useful information you may need to answer:
    {RELEVANT-TABLES-DESCRIPTION}
"""
```

```
# Step 4: Validate the generated query
"""
```

```
    Given the following database schema:
    {DB-SCHEMA}
    {INSERT-SAMPLES}
    Questions: {REPHRASED-QUESTIONS}
    Query: {GENERATED-QUERY}
    Tell me if the query answers all these equivalent questions
   correctly. Answer only with YES or NO.
"""
```

Bibliography

[1] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL]. URL: https://arxiv.org/abs/1409.3215 (cit. on pp. 2, 9).

[2] Sepp Hochreiter and Jürgen Schmidhuber. «Long short-term memory». In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on pp. 2, 8).

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805 (cit. on pp. 2, 10).

[4] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.LG]. URL: https://arxiv.org/abs/1910.10683 (cit. on pp. 2, 10).

[5] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165 (cit. on pp. 2, 10, 22).

[6] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774 (cit. on pp. 2, 10, 13).

[7] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. *Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation*. 2023. arXiv: 2308.15363 [cs.DB]. URL: https://arxiv.org/abs/2308.15363 (cit. on pp. 4, 16, 23, 25, 30).

[8] Joseph Weizenbaum. «ELIZA—a computer program for the study of natural language communication between man and machine». In: *Commun. ACM* 9.1 (Jan. 1966), pp. 36–45. ISSN: 0001-0782. DOI: 10.1145/365153.365168. URL: https://doi.org/10.1145/365153.365168 (cit. on p. 7).

[9] Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer. «Class-Based *n*-gram Models of Natural Language». In: *Computational Linguistics* 18.4 (1992), pp. 467–480. URL: https://aclanthology.org/J92-4003/ (cit. on p. 7).

[10] Alex Sherstinsky. «Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network». In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020), p. 132306. ISSN: 0167-2789. DOI: 10.1016/j.physd.2019.132306. URL: http://dx.doi.org/10.1016/j.physd.2019.132306 (cit. on p. 7).

[11] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. arXiv: 1912.05911 [cs.LG]. URL: https://arxiv.org/abs/1912.05911 (cit. on p. 7).

[12] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781 (cit. on pp. 8, 9).

[13] Jeffrey Pennington, Richard Socher, and Christopher Manning. «GloVe: Global Vectors for Word Representation». In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162/ (cit. on p. 9).

[14] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. *Enriching Word Vectors with Subword Information*. 2017. arXiv: 1607.04606 [cs.CL]. URL: https://arxiv.org/abs/1607.04606 (cit. on p. 9).

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762 (cit. on p. 9).

[16] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL]. URL: https://arxiv.org/abs/1910.13461 (cit. on p. 10).

[17] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401 (cit. on p. 12).

[18] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361 (cit. on p. 12).

[19] OpenAI. *OpenAI Official Website*. Accessed: Mar-2025. 2025. URL: https://openai.com/ (cit. on p. 13).

[20] Gemini Team et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL]. URL: https://arxiv.org/abs/2312.11805 (cit. on p. 13).

[21] Gemini Team et al. *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. 2024. arXiv: 2403.05530 [cs.CL]. URL: https://arxiv.org/abs/2403.05530 (cit. on p. 13).

[22] Google DeepMind. *Google DeepMind Official Website*. Accessed: Mar-2025. 2025. URL: https://deepmind.google/ (cit. on p. 13).

[23] Anthropic et al. *The Claude 3 Model Family: Opus, Sonnet, Haiku*. March 2024. URL: https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf (cit. on p. 13).

[24] Anthropic. *Anthropic Official Website*. Accessed: Mar-2025. 2025. URL: https://www.anthropic.com/ (cit. on p. 13).

[25] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971 (cit. on p. 13).

[26] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL]. URL: https://arxiv.org/abs/2307.09288 (cit. on p. 13).

[27] Aaron Grattafiori et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783 (cit. on p. 13).

[28] Albert Q. Jiang et al. *Mistral 7B*. 2023. arXiv: 2310.06825 [cs.CL]. URL: https://arxiv.org/abs/2310.06825 (cit. on p. 13).

[29] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. arXiv: 2501.12948 [cs.CL]. URL: https://arxiv.org/abs/2501.12948 (cit. on pp. 13, 68).

[30] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. *Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL*. 2024. arXiv: 2406.08426 [cs.CL]. URL: https://arxiv.org/abs/2406.08426 (cit. on p. 17).

[31] Prashanth Gurunath Shivakumar, Mu Yang, and Panayiotis Georgiou. «Spoken Language Intent Detection Using Confusion2Vec». In: *Interspeech 2019*. interspeech$_2$019. ISCA, Sept. 2019, pp. 819–823. DOI: 10.21437/interspeech.2019-2226. URL: http://dx.doi.org/10.21437/Interspeech.2019-2226 (cit. on p. 18).

[32] IMDb. *IMDb Dataset*. Accessed: Mar-2025. 2025. URL: https://developer.imdb.com/non-commercial-datasets/ (cit. on p. 18).

[33] Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. *FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis*. 2024. arXiv: 2401.10506 [cs.CL]. URL: https://arxiv.org/abs/2401.10506 (cit. on p. 18).

[34] Victor Zhong, Caiming Xiong, and Richard Socher. *Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning*. 2017. arXiv: 1709.00103 [cs.CL]. URL: https://arxiv.org/abs/1709.00103 (cit. on p. 18).

[35] Tao Yu et al. *Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task*. 2019. arXiv: 1809.08887 [cs.CL]. URL: https://arxiv.org/abs/1809.08887 (cit. on pp. 18, 19).

[36] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. «Structure-Grounded Pretraining for Text-to-SQL». In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2021. DOI: 10.18653/v1/2021.naacl-main.105. URL: http://dx.doi.org/10.18653/v1/2021.naacl-main.105 (cit. on p. 18).

[37] Yujian Gan, Xinyun Chen, and Matthew Purver. *Exploring Underexplored Limitations of Cross-Domain Text-to-SQL Generalization*. 2021. arXiv: 2109.05157 [cs.CL]. URL: https://arxiv.org/abs/2109.05157 (cit. on p. 18).

[38] Jinyang Li et. al. *Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs*. 2023. arXiv: 2305.03111 [cs.CL]. URL: https://arxiv.org/abs/2305.03111 (cit. on pp. 18, 20).

[39] Tao Yu et al. *SParC: Cross-Domain Semantic Parsing in Context*. 2019. arXiv: 1906.02285 [cs.CL]. URL: https://arxiv.org/abs/1906.02285 (cit. on p. 19).

[40] Tao Yu et al. *CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases*. 2019. arXiv: 1909.05378 [cs.CL]. URL: https://arxiv.org/abs/1909.05378 (cit. on p. 19).

[41] Jiaqi et al. Guo. «Chase: A Large-Scale and Pragmatic Chinese Dataset for Cross-Database Context-Dependent Text-to-SQL». In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Ed. by Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli. Online: Association for Computational Linguistics, Aug. 2021, pp. 2316–2331. DOI: 10.18653/v1/2021.acl-long.180. URL: https://aclanthology.org/2021.acl-long.180/ (cit. on p. 19).

[42] Ruiqi Zhong, Tao Yu, and Dan Klein. *Semantic Evaluation for Text-to-SQL with Distilled Test Suites*. 2020. arXiv: 2010.02840 [cs.CL]. URL: https://arxiv.org/abs/2010.02840 (cit. on p. 20).

[43] Qingxiu Dong et al. *A Survey on In-context Learning*. 2024. arXiv: 2301.00234 [cs.CL]. URL: https://arxiv.org/abs/2301.00234 (cit. on p. 22).

[44] Mohammadreza Pourreza and Davood Rafiei. *DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction*. 2023. arXiv: 2304.11015 [cs.CL]. URL: https://arxiv.org/abs/2304.11015 (cit. on p. 23).

[45] Haoyang Li et al. *CodeS: Towards Building Open-source Language Models for Text-to-SQL*. 2024. arXiv: 2402.16347 [cs.CL]. URL: https://arxiv.org/abs/2402.16347 (cit. on p. 23).

[46] Zhishuai Li et al. *PET-SQL: A Prompt-Enhanced Two-Round Refinement of Text-to-SQL with Cross-consistency*. 2024. arXiv: 2403.09732 [cs.CL]. URL: https://arxiv.org/abs/2403.09732 (cit. on pp. 23, 26).

[47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL]. URL: https://arxiv.org/abs/2201.11903 (cit. on p. 23).

[48] Denny Zhou et al. «Least-to-Most Prompting Enables Complex Reasoning in Large Language Models». In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=WZH7099tgfM (cit. on p. 23).

[49] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. *C3: Zero-shot Text-to-SQL with ChatGPT*. 2023. arXiv: 2307.07306 [cs.CL]. URL: https://arxiv.org/abs/2307.07306 (cit. on p. 23).

[50] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. *CHESS: Contextual Harnessing for Efficient SQL Synthesis*. 2024. arXiv: 2405.16755 [cs.LG]. URL: https://arxiv.org/abs/2405.16755 (cit. on p. 23).

[51] Bing Wang et al. *MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL*. 2025. arXiv: 2312.11242 [cs.CL]. URL: https://arxiv.org/abs/2312.11242 (cit. on p. 23).

[52] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. *Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation*. 2019. arXiv: 1905.08205 [cs.CL]. URL: https://arxiv.org/abs/1905.08205 (cit. on p. 24).

[53] Ursin Brunner and Kurt Stockinger. *ValueNet: A Natural Language-to-SQL System that Learns from Database Information*. 2021. arXiv: 2006.00888 [cs.DB]. URL: https://arxiv.org/abs/2006.00888 (cit. on p. 24).

[54] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. *RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers*. 2021. arXiv: 1911.04942 [cs.CL]. URL: https://arxiv.org/abs/1911.04942 (cit. on p. 24).

[55] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. *RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL*. 2023. arXiv: 2302.05965 [cs.CL]. URL: https://arxiv.org/abs/2302.05965 (cit. on p. 24).

[56] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. 2023. arXiv: 2203.11171 [cs.CL]. URL: https://arxiv.org/abs/2203.11171 (cit. on p. 24).

[57] Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. *Automatically Correcting Large Language Models: Surveying the landscape of diverse self-correction strategies*. 2023. arXiv: 2308.03188 [cs.CL]. URL: https://arxiv.org/abs/2308.03188 (cit. on p. 25).

[58] Ke Shen and Mayank Kejriwal. *SelECT-SQL: Self-correcting ensemble Chain-of-Thought for Text-to-SQL*. 2024. arXiv: 2409.10007 [cs.CL]. URL: https://arxiv.org/abs/2409.10007 (cit. on p. 25).

[59] Fangyu Lei et al. *Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows*. 2025. arXiv: 2411.07763 [cs.CL]. URL: https://arxiv.org/abs/2411.07763 (cit. on p. 25).

[60] Yingqi Gao et al. *A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL*. 2025. arXiv: 2411.08599 [cs.AI]. URL: https://arxiv.org/abs/2411.08599 (cit. on p. 25).

[61] Google Cloud. *Gemini Models Documentation*. Accessed: Mar-2025. URL: https://cloud.google.com/vertex-ai/generative-ai/docs/learn/models (cit. on p. 29).

[62] Asma Ben Abacha, Wen-wai Yim, Yujuan Fu, Zhaoyi Sun, Meliha Yetisgen, Fei Xia, and Thomas Lin. *MEDEC: A Benchmark for Medical Error Detection and Correction in Clinical Notes*. 2025. arXiv: 2412.19260 [cs.CL]. URL: https://arxiv.org/abs/2412.19260 (cit. on p. 29).