Tuning Large Language Models for Text-to-SQL on the IUPHAR/BPS Guide to Pharmacology

Nikita Rameshkumar

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2025

Abstract

This project explores the integration of OpenAI's GPT-4o Large Language Model (LLM) with the IUPHAR/BPS Guide to Pharmacology database to enable natural language queries (NLQs) for retrieving pharmacological data through SQL conversion. Several strategies were explored, culminating in a four-component prompt that incorporates schema structure, many-shot learning using NLQ-SQL pairs, manually written rules, and self-correction for error handling. The effectiveness of the prompt was evaluated on a separate test set using metrics such as successful execution rate, non-empty output rate, execution accuracy, and partial execution accuracy. The results were compared to an alternative implementation approach across all metrics. Additionally, an analysis of token usage and associated costs was conducted.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, that this work has not been submitted for any other degree or professional qualification except as specified, and that this is a modified version of the thesis originally submitted.

(*Nikita Rameshkumar*)

Acknowledgements

I would like to express my sincere gratitude to David Sterratt, Simon Harding, Jamie Davies, the IUPHAR Group and Alexandra Birch-Mayne for their invaluable support, insightful feedback, and guidance throughout this project. I am also deeply thankful to Ian Little for working alongside me on this project.

I am especially grateful to my family, who have been my pillars of support in every sense. Their unwavering love, encouragement, and sacrifices have shaped my journey, and I will always be grateful that they moved to this country for me, standing by my side through every challenge and success–none of which would have been possible without them.

Finally, I extend my appreciation to the University of Edinburgh and this beautiful city for providing me with the opportunity to expand my knowledge and grow both academically and personally through these years, and to the many incredible people I have met along the way whose friendships and support have made this experience all the more meaningful.

Contents

Chapter 1

Introduction

Since the release of ChatGPT by OpenAI in November 2022, Large Language Models (LLMs) have gained widespread popularity (Meyer et al., 2023). This shift has changed how users interact with information retrieval systems, including search engines, databases, and the broader internet (Tabarsi et al., 2025). To keep up with these changes, integrating LLMs into information retrieval has become increasingly important, ensuring users can access data effectively (Weber, 2024). The IUPHAR/BPS Guide to Pharmacology database (GtoPdb), curated from expert literature, currently requires SQL for querying its dataset. This dissertation explores how OpenAI's LLMs can enable natural language access to GtoPdb.

1.1 Aims and Objectives

This project aims to enhance access to the IUPHAR/BPS Guide to Pharmacology (GtoPdb) by using OpenAI's Large Language Models to convert natural language queries into SQL, eliminating the need for users to know SQL or the database structure. LLMs have outperformed traditional text-to-SQL methods in benchmark datasets, suggesting that OpenAI's LLMs can provide a more efficient solution (Hong et al., 2024). However, since OpenAI's LLM is trained on general data, it lacks specific knowledge of GtoPdb's structure. Therefore, the model will be fine-tuned to handle the GtoPdb schema and generate accurate SQL queries. Enabling natural language queries has the potential to make GtoPdb more accessible to its audience of pharmacology researchers and students, who may not have knowledge of writing SQL.

A key objective is to fine-tune the LLM using relevant data and test it on a set of 30 natural language queries (NLQs) and their corresponding gold standard SQL queries curated by the NC-IUPHAR Database Executive Committee [1]. Model performance will be evaluated using four metrics: Successful Execution Rate (SER), Non-Empty Output Rate (NER), Execution Accuracy (EX), and Partial Execution Accuracy (PEX). The goal is to achieve the highest possible scores on these metrics, with an SER above 90% and an NER above 80%, ensuring that the fine-tuned LLM generates executable SQL

[1] IUPHAR: `https://iuphar.org/`

queries that match the expected results from the gold standard SQL in the test set.

Another key objective is to develop a pipeline that integrates the fine-tuned LLM with a backend system. Users will input natural language queries, which the tuned LLM will convert into SQL queries, executed on the database to retrieve results. It is essential to define the boundaries for implementing the pipeline, ensuring that the LLM provides factually accurate and up-to-date information exclusively based on the contents of the database, without any manipulation or hallucination. Additionally, the system must be designed to prevent the generation or dissemination of any harmful or misleading information.

1.2 Overview

Chapter 2 introduces Large Language Models (LLMs), outlining their fundamental principles and potential challenges. It explores the application of LLMs in text-to-SQL tasks, particularly how they can be enhanced with a knowledge base to provide more specific and dynamic information. Additionally, it discusses the relative performance of different LLMs on text-to-SQL tasks and the key metrics used to evaluate them, with a particular focus on the BIRD benchmark and dataset. This is followed by a comparative analysis of available models to determine the most suitable choice for this task. The chapter also introduces prompt engineering, defining its role in optimising model performance, and presents criteria for effective prompt design. Finally, an overview of the Guide to Pharmacology database is provided, detailing its relevance to this project.

Chapter 3 presents the system architecture and the overall workflow, describing the structure of the implemented pipeline and the data flow from input to output. It then provides details on the NLQ-SQL dataset, curated by the NC-IUPHAR Database Executive Committee, which was used for model tuning and testing. The chapter further delves into the prompt implementation strategy, detailing its four key components: GtoPdb schema; many-shot learning; refinement rules; and self-correction. Additionally, the testing methodology is outlined, introducing a specialised test suite with four evaluation metrics: Successful Execution Rate, Non-Empty Output Rate, Execution Accuracy, and Partial Execution Accuracy.

Chapter 4 presents the experimental process undertaken to optimise the prompt and identify the most effective configuration. This includes an iterative exploration of different many-shot learning strategies and an ablation study on various prompt components. The chapter discusses the final prompt's performance on both the development and test sets, providing a comparative analysis of the results. Furthermore, it examines an alternative pipeline and prompt engineering strategy developed by another undergraduate student, comparing its test set performance against the proposed approach. Finally, details on LLM runtime, token usage, and associated computational costs are analysed.

Chapter 5 summarises the key findings of this project, highlighting the developed system and its achieved results. It also outlines potential areas for future work, focusing on enhancements to both the methodology and overall system performance.

Chapter 2

Background

2.1 Large Language Models

A machine learning model is a type of mathematical model that, once trained on a given dataset, can be used to make predictions or classifications on new data. Large language models (LLMs) are machine learning models that apply neural network techniques with millions of parameters to process human languages or text through prompts using self-supervised learning techniques. This makes LLMs capable of generating human-like text and allows users to interact with them using natural language (Naveed et al., 2023). LLMs are accessible through interfaces for conversational capabilities like Open AI's Chat GPT-3 and GPT-4 [1]. Other examples include Meta's Llama models [2] and Google's bidirectional encoder representations from transformers (BERT/RoBERTa) and PaLM models [3]. This project specifically focuses on the use of OpenAI's GPT-4o model, chosen for its optimal balance between performance and cost, as detailed in Subsection 2.1.2.

Currently, LLMs are pre-trained on vast amounts of data, which means they lack up-to-date information on dynamic databases that are constantly evolving. LLMs only "know" the data they were trained on, which is typically limited to what was available on the internet up until the time of their training. While these models are highly complex, they remain static in their knowledge (Yildirim and Paul, 2024; Matarazzo and Torlone, 2025).

In cases where the knowledge of current or dynamic data is required, an LLM would require access to the dynamic knowledge base and the ability to be specifically instructed on new tasks or information to retrieve. Moreover, LLMs have other limitations, such as their tendency to generate plausible yet factually incorrect outputs, often referred to as "hallucinations" (Matarazzo and Torlone, 2025). Ultimately, LLMs are constrained by the patterns they learned during training (Matarazzo and Torlone, 2025). LLMs also tend to provide varying outputs for the same user prompt, which should be considered for

[1] OpenAI GPT-4: `https://openai.com/index/gpt-4/`

[2] Meta Llama 3.2: `https://www.llama.com/`

[3] Palm 2: `https://ai.google/discover/palm2/`

LLM tuning and evaluation (Shen et al., 2025). This highlights a critical objective: the GtoPdb LLM must not only be factually accurate but also rely on an external knowledge base of the database structure to ensure reliable and effective query generation (Truhn et al., 2023).

2.1.1 LLMs for Text-to-SQL

In recent years, text-to-SQL parsing, which converts natural language queries into executable SQL statements, has gained significant attention. Models such as GPT-4 and Claude-2 have shown leading performance, with execution accuracy rates above 50%, setting a new benchmark in the field (Li et al., 2023). In contrast to text-to-SQL benchmarks such as Spider [4] and WikiSQL [5], the BIRD benchmark captures the complexity of real-world applications, which includes 12,751 text-to-SQL pairs (11,218 training set and 1,533 development set) and 95 large-scale databases across 37 professional domains (Li et al., 2023; Wretblad and Gordh Riseby, 2024).

This aligns with the challenges posed by the Guide to Pharmacology, which aims to generate SQL queries on a specialized scientific database. State-of-the-art systems like GPT-4, achieve an execution accuracy of 54.89% (with human performance of EX 92.96%) (Li et al., 2023) which we will use as the benchmark for this project. The metrics used to evaluate models on the BIRD benchmark are Execution Accuracy (EX), which evaluates the content returned by the query and Valid Efficiency Score (VES), which evaluates the efficiency of the generated SQL query. A comparison of EX and VES across different LLMs is summarised in Figure 2.1.

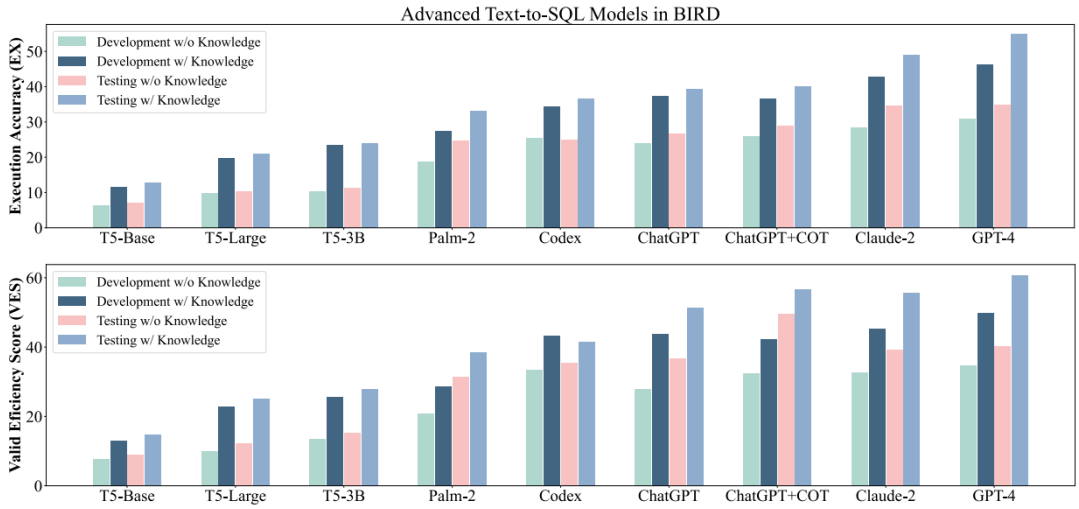

Figure 2.1: Comparison of EX and VES across different LLMs on the BIRD dataset (Li et al., 2023)

Figure 2.1 shows that the top-performing models are generally from OpenAI, with GPT-4 achieving the best results and Claude-2 being the only exception to this trend. However, with the rapid advancement of these models, GPT-4o has now replaced GPT-4

[4]Spider: `https://yale-lily.github.io/spider`
[5]WikiSQL: `https://paperswithcode.com/dataset/wikisql`

as OpenAI's flagship model, offering enhanced capabilities. Additionally, we will evaluate performance using Execution Accuracy (EX) as what was used in BIRD, which is defined as the proportion of examples in the evaluation set for which the executed results of both the predicted and ground-truth SQLs are identical, relative to the total number of SQLs. Considering the result set V_n executed by the n-th ground-truth SQL Y_n, and the result set $\hat{V}_n$ executed by the predicted SQL $\hat{Y}_n$, EX is computed as follows:

$$EX = \frac{\sum_{n=1}^{N} \mathbf{1}(V_n, \hat{V}_n)}{N}$$

where $\mathbf{1}(V, \hat{V})$ is an indicator function that is defined as:

$$\mathbf{1}(V, \hat{V}) = \begin{cases} 1, & \text{if } V = \hat{V} \\ 0, & \text{if } V \neq \hat{V} \end{cases}$$

(Li et al., 2023)

We will not focus on efficiency, so the VES metric will be excluded from our evaluation. Our primary interest is in the quality of content returned by the tuned LLM, and we will gather more in-depth metrics in terms of EX, along with related data. Since LLMs can improve by learning from their failures through trial-and-error methods, self-correction can be incorporated (Pourreza and Rafiei, 2023). This would further enhance text-to-SQL applications by enabling the model to refine its queries and recover from failed attempts (Pourreza and Rafiei, 2023).

2.1.2 Model Comparison and Selection

As shown in Figure 2.1, GPT-4 was previously the leading model for text-to-SQL tasks. However, it has a limited 8k context window, which restricts its ability to handle complex databases like the Guide to Pharmacology. Additionally, newer models offer enhanced capabilities and better performance. For this project, GPT-4o was chosen for its 128k context length, cost-efficiency, and ability to be fine-tuned and function implementation to meet specific requirements. It outperforms GPT-4 in both context size and cost-effectiveness, making it more suitable for the task. Reasoning models, such as `o1` and `o3-mini`, offer larger context windows (200k tokens), but they do not support fine-tuning and were therefore excluded. GPT-4.5 and GPT-4o-mini were also considered, but GPT-4o was chosen for its balance of performance, cost, and fine-tuning support. Notably, fine-tuning these LLMs incurs double the input cost. Table 2.1 [6] summarises the context lengths and pricing of the considered OpenAI models.

2.2 Prompt Engineering

The effectiveness of the LLM depends on how precise the instructions conveyed to it are (Nan et al., 2023). This process of formulating instructions in natural language

[6] OpenAI Pricing: `https://platform.openai.com/docs/pricing`

Model	Context Length	Input Cost	Cached Input Cost	Output Cost
GPT-4	8k	30.00	-	60.00
GPT-4.5	128k	75.00	37.50	150.00
GPT-4o	128k	2.50	1.25	10.00
GPT-4o-mini	128k	0.15	0.07	0.60
o1	200k	15.00	7.50	60.00
o3-mini	200k	1.10	0.55	4.40

Table 2.1: OpenAI Pricing (Costs in $ per 1M tokens)

within a prompt is referred to as prompt engineering. A well-structured prompt reduces ambiguity, enforces the correct output format, and aligns the model's response with the task objectives (Drushchak et al., 2024). Figure 2.2 illustrates these different components of an ideal prompt.

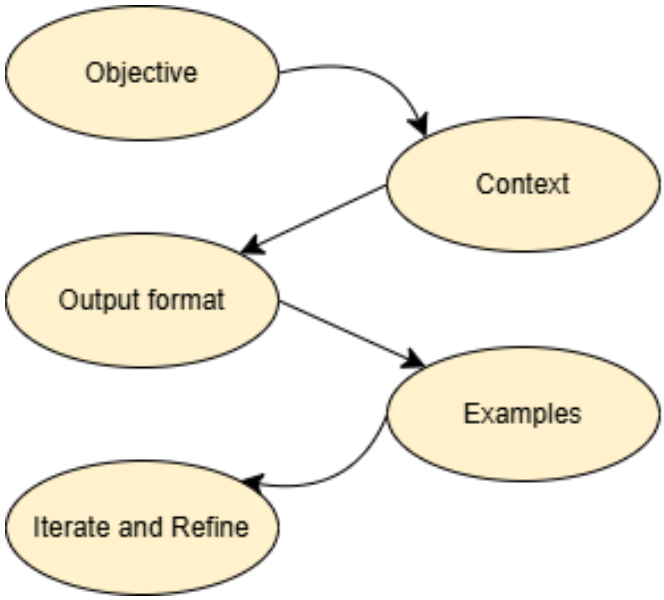

Figure 2.2: Components of an Ideal Prompt

As shown in Figure 2.2, an effective prompt for text-to-SQL applications should clearly specify the LLM's objective–translating natural language queries (NLQs) into executable SQL commands. In the context of this project, providing relevant information entails supplying the LLM with the GtoPdb schema and any additional details essential for accurate query generation (Li et al., 2023). The prompt should also define the expected output format (syntactically correct and executable SQL), and include examples of NLQ-SQL pairs (few-shot learning) to guide the model's performance. Iterative refinement of the prompt, based on output evaluation, further improves performance. This process ensures the LLM generates accurate and executable SQL aligned with the task requirements (Nan et al., 2023; Chang and Fosler-Lussier, 2023).

In OpenAI's API, prompt engineering starts with a system message that defines the model's behaviour before any user interaction. For example, a simple system message could be: "You are a Guide to Pharmacology expert that converts natural language

queries to SQL queries." This message is set during the initial API call, and the tokens used are counted as context tokens. In this project, the system message is prompt engineered as part of the development and tuning process.

2.3 The IUPHAR/BPS Guide to Pharmacology

The International Union of Basic and Clinical Pharmacology (IUPHAR) / British Pharmacological Society (BPS) Guide to PHARMACOLOGY is an expert-curated database consisting of 224 tables of ligand-activity-target relationships, sourced from high-quality pharmacological and medicinal chemistry literature (Harding et al., 2024). It is used extensively in drug discovery, basic and clinical research, and education. GtoPdb primarily serves pharmacologists in research and students studying pharmacology. It is important to consider these users needs and expectations when designing the LLM, ultimately aiming to make the database more accessible.

The integration of LLMs with the GtoPdb enhances accessibility to complex pharmacological data for researchers. By utilising LLMs, users can query the database using natural language, enabling them to obtain information without requiring deep technical expertise in pharmacology or database management. LLMs can interpret user queries that may vary in complexity and terminology, translating them into precise database queries. For example, a researcher might ask, "What are the side effects of Drug X?" or "Find all approved drugs". The LLM would process this query, identify relevant keywords, and retrieve the corresponding information from GtoPdb. This not only enhances the user experience but streamlines the research process, allowing for quicker insights into drug interactions, mechanisms of action, and efficacy. Furthermore, LLMs can enhance education by providing explanations and contextual information about pharmacological concepts and drug data, enabling students and researchers to deepen their understanding of pharmacology through an intuitive dialogue-based interface with GtoPdb.

It is also crucial to establish boundaries to prevent the system from providing harmful information or taking actions that could pose risks while delivering drug-related insights. The information provided by the tool should be accurate and not be misleading about drug interactions, dosages, or side effects. This can impact researchers or healthcare professionals, potentially leading to inappropriate treatment decisions. The information provided by the tool should also be up to date and reflect the current guidelines and findings related to drugs. The user should also only be allowed to carry out 'SELECT' SQL queries through the LLM and not be able to insert or delete information from the database which could cause the depiction of incorrect information or corrupt the data. Lastly, users should not be allowed to ask for medical advice from the database.

Chapter 3

Methodology

This section provides an overview of the system, detailing the pipeline design, the implementation strategies employed to address the challenges of this dissertation, and the testing methodologies used to evaluate the LLM's performance.

3.1 System Overview

The system comprises multiple components designed to tune the GPT-4o LLM according to specific requirements, process natural language queries, and present accurate results to the user. To ensure that the system reliably displays only factual information retrieved from the database, a robust pipeline consisting of various components was developed. The system relies on the OpenAI API for LLM queries and therefore necessitates an API key to run. Figure 3.1 provides an overview of the system that was built for this project.

The flow of actions illustrated in Figure 3.1 is initiated when a user submits a natural language query (NLQ). This query is sent to the tuned LLM. The LLM is dynamically tuned during interaction and is not a pre-existing fine-tuned model, due to its reliance on the OpenAI API. The system message for the LLM is set as a constant, and when the NLQ is made, it is sent as part of the message payload. API calls are made with the context tokens, followed by the user message, which is processed by the LLM. Because the LLM's context has been prompt-engineered in advance, it generates an SQL query based on the user's input. The GtoPdb database is installed locally for development purposes, using PostgreSQL as the database software and psycopg2 [1], a Python-PostgreSQL Database Adapter for database connection. Once the connection is established, the number of retries is initially set to 0 and the SQL query generated by the LLM is executed on the database.

If an error occurs during query execution, such as an SQL syntax error, incorrect column names, or if the query executes but returns an empty dataframe, the NLQ is retried. In this retry, the previous error message is included in the context; the new user query becomes the original NLQ appended with a descriptive error message. The retry count

[1] psycopg2: `https://pypi.org/project/psycopg2/`

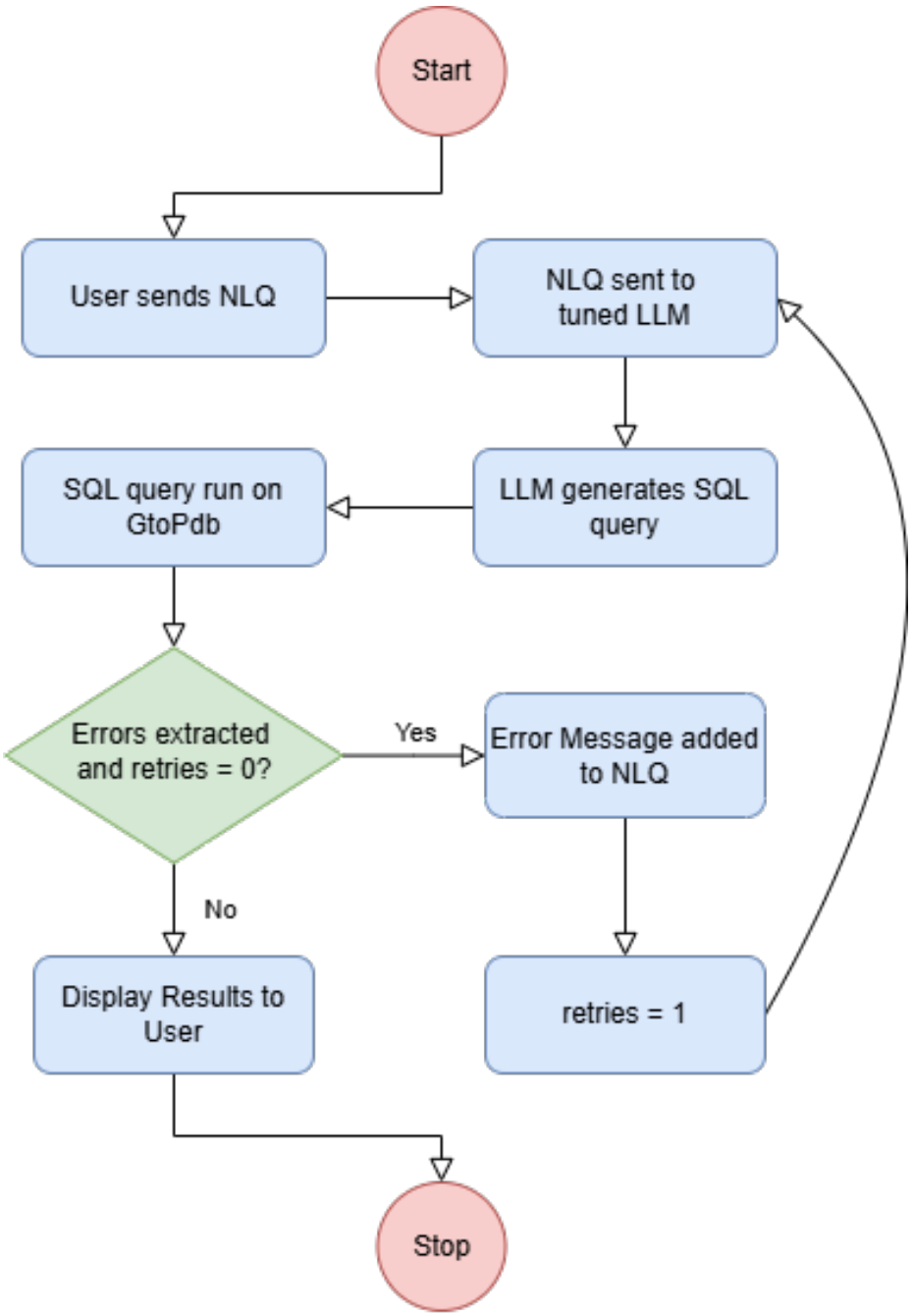

Figure 3.1: GtoPdb LLM System Flowchart

is then set to 1, and this modified query is sent again to the tuned LLM. This retry mechanism called self-correction, is only triggered if the number of retries is 0 and an error previously occurred. If a retry has already been performed once, or if no errors are encountered in the initial execution, the system proceeds to display whatever result was returned-whether it is valid data or an empty result. This result is presented to the user in the form of a table. This marks the end of the process for handling a single natural language query, from submission to retrieving and displaying results from the database. By not routing the results through the LLM, the system ensures that the data remains unaltered, and the user is presented with the direct output of the executed SQL query.

However, this project focuses solely on the backend implementation of the pipeline and the tuning of the LLM. As such, no user interface has been implemented, and it is assumed that the results will ultimately be delivered to the user.

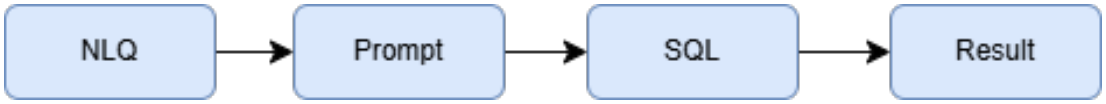

Figure 3.2: NLQ to Result Execution Flow

Figure 3.2 illustrates the process of converting a natural language query into its corresponding SQL result, as outlined in the pipeline overview. The query is first transformed into a structured prompt, which is then used to generate an SQL query. This query is subsequently executed, producing the final result. Appendix A highlights this process through an example.

3.2 NLQ-SQL Dataset

A dataset comprising a total of 81 queries was curated by the NC-IUPHAR Database Executive Committee to support the training and evaluation of the pharmacology-focused large language model (LLM). These natural language queries (NLQs) were then manually translated into corresponding SQL queries. The dataset was divided into a training set and a test set. The training set consists of 51 NLQ-SQL pairs, which were used throughout the development phase of the system. The test set includes the remaining 30 NLQ-SQL pairs, which were reserved for final evaluation after development was complete. To ensure an unbiased assessment, the test set was withheld until the conclusion of the development phase. Appendix C and Appendix D present some NLQs from the development and test sets.

Each query was classified by difficulty, using one of four levels: Easy, Easy-Moderate, Moderate-Hard, and Hard. When splitting the data into training and test sets, stratified sampling was applied to maintain approximately equal proportions of each difficulty level in both sets. In cases where queries were considered similar, care was taken to allocate them to different splits to ensure a diverse and representative evaluation. Some natural language queries in the dataset were considered inherently vague, with no definitive correct answer. In such cases, the provided SQL query represents one example of a valid interpretation or reasonable answer. The dataset includes the following metadata columns for each NLQ-SQL pair:

- **ID**: The identifier for the query in the complete dataset.
- **Difficulty**: One of the four values: *Easy*, *Easy-Moderate*, *Moderate-Hard*, *Hard*, where a value of ‘1‘ indicates the assigned difficulty level.
- **Greek**: Indicates whether the natural language query contains a Greek letter.
- **Vague/No definite right answer**: Marks queries where the expected output is likely to be vague.

- **Minimum output columns**: Specifies the minimum number of columns that should be retrieved to answer the query. While most queries require only one column, including additional columns may improve usability for the end user.
- **Notes for student**: Contains notes on relevant aspects of the database schema
- **Training/test set**: Indicates whether the item belongs to the training set or the test set.
- **SQL**: The SQL query corresponding to the natural language query.
- **2nd SQL**: An alternative SQL query (provided for some queries).

Two examples from the training set are provided below to depict the format and variation in complexity of the NLQ-SQL pairs.

1. Natural Language Query (Difficulty = Easy): *"Find all approved drugs"*

The LLM must infer that a "drug" refers to a ligand, retrieve relevant information from the corresponding table, and apply conditions based on the column names.

```
SELECT ligand_id, name
FROM ligand
WHERE approved IS TRUE;
```

2. Natural Language Query (Difficulty = Hard): *"What ligands and structures have pKi values between 5 and 7 and are the primary targets of a GPCR?"*

```
SELECT l.ligand_id, l.name, l.type, ls.isomeric_smiles, ls.
   isomeric_standard_inchi_key, i.type, i.action,
       CASE WHEN i.affinity_median IS NULL
            THEN CASE WHEN i.affinity_high IS NULL
                      THEN i.affinity_low
                      ELSE i.affinity_high
                 END
            ELSE i.affinity_median
       END AS affinity,
       i.affinity_units
  FROM ligand l, interaction i
       JOIN ligand_structure ls ON i.ligand_id = ls.ligand_id
 WHERE i.object_id IN (SELECT object_id FROM gpcr)
   AND i.ligand_id = l.ligand_id
   AND i.primary_target IS TRUE
   AND CASE WHEN i.affinity_median IS NULL
            THEN CASE WHEN i.affinity_high IS NULL
                      THEN i.affinity_low BETWEEN 5 AND 7
                      ELSE i.affinity_high BETWEEN 5 AND 7
                 END
            ELSE i.affinity_median BETWEEN 5 AND 7
       END
ORDER BY affinity DESC;
```

3.3 Prompt Implementation Strategy

Various forms of input were used to prompt the LLM during the prompt engineering process. This process involves training the LLM with natural language, requiring the design of a prompt that is both concise and descriptive. My approach was to provide the LLM with as much relevant information as possible. Before sending the natural language query to the LLM, it is first prompted with context to guide its function. As shown in Figure 2.2, an ideal prompt should include the objective, context, output format, examples, and an iteration and refinement process. The goal was to incorporate all of these aspects into the prompt.

The prompt begins with the statement: "You are an IUPHAR Guide to Pharmacology expert that converts natural language queries to SQL," which establishes the LLM's objective. Next, the LLM was provided with the full GtoPdb schema/structure in the form of a JSON as context. Following this, many-shot learning was used by providing examples from the complete training dataset of 51 NLQ-SQL pairs. For the iteration and refinement component, the prompt was refined by adjusting a set of manually written rules based on the LLM's performance on the training set. Finally, the fourth component involved including any relevant previous error messages for the same NLQ, known as self-correction. Figure 3.3 illustrates the four key components of the prompt used to fine-tune the GPT-4o LLM.

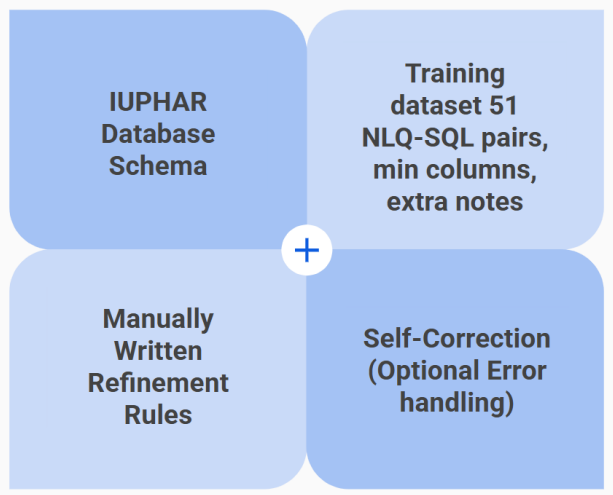

Figure 3.3: Components of the Text-to-SQL system prompt

These four components equipped the LLM with the necessary context to accurately convert user NLQs into SQL queries.

3.3.1 Database Schema

The GtoPdb schema was stored in a JSON file, which served as a mapping between table names and their respective column names. This structure ensured that the LLM had complete knowledge of all available tables and their columns. However, the schema did not include data types for the columns, as doing so would significantly increase token usage. Specifically, while table names required 224 tokens and column names accounted

for 1823 tokens (a total of 2047 tokens), including data types would have increased this to 3870 tokens per user query. To minimise token consumption, we omitted data types, relying instead on the LLM's ability to infer them based on column names. A sample representation of two tables, `accessory_protein` and `xenobiotic_expression_refs`, from the GtoPdb, as stored in the JSON file, is shown below:

```
{
    "tables": [
        {
            "table_name": "accessory_protein",
            "columns": ["object_id", "full_name"]
        },
        {
            "table_name": "xenobiotic_expression_refs",
            "columns": ["xenobiotic_expression_id", "
               reference_id"]
        }
    ]
}
```

To integrate this schema into the prompt, it needed to be represented as a string. The JSON structure was therefore converted into a string format for compatibility. The schema was introduced in the prompt using the following format:

> **You have the following PostgreSQL database schema. The database consists of the following tables:**
> - **Table:** `accessory_protein`
> - **Columns:** `object_id, full_name`
> - **Table:** `xenobiotic_expression_refs`
> - **Columns:** `xenobiotic_expression_id, reference_id` ...

The total token usage for this section of the prompt was calculated as 2495 tokens as depicted in Table 3.1. Given that GPT-4o costs $5 per 1M tokens for context tokens, this portion of the prompt incurs a cost of approximately $0.0125 per user query, i.e., per OpenAI API call. This prompting method ensures that the LLM has full knowledge of all the tables and their corresponding column names in the GtoPdb.

Component	**Token Usage**
Table names	224
Column names	1823
"Table:" mentions	224
"Columns:" mentions	224
Total	2495
Cost	$0.0125

Table 3.1: Token usage breakdown for the database schema prompt

3.3.2 Many-Shot prompting

Few-shot prompting refers to a technique in which an LLM is given a small number of examples (typically 1 to 5) to guide its response generation. In contrast, many-shot learning involves providing a much larger number of examples–ranging from 50 to hundreds or even thousands–allowing the LLM to learn more effectively from patterns in the data. Both approaches fall under the broader paradigm of in-context learning, where the LLM learns from question-answer examples provided in the prompt. Research has shown that many-shot learning significantly improves performance compared to few-shot prompting (Agarwal et al., 2024). Figure 3.4 illustrates a comparison between few-shot and many-shot tuning of the Gemini 1.5 Pro LLM across 11 tasks, demonstrating that many-shot learning consistently outperforms the few-shot approach.

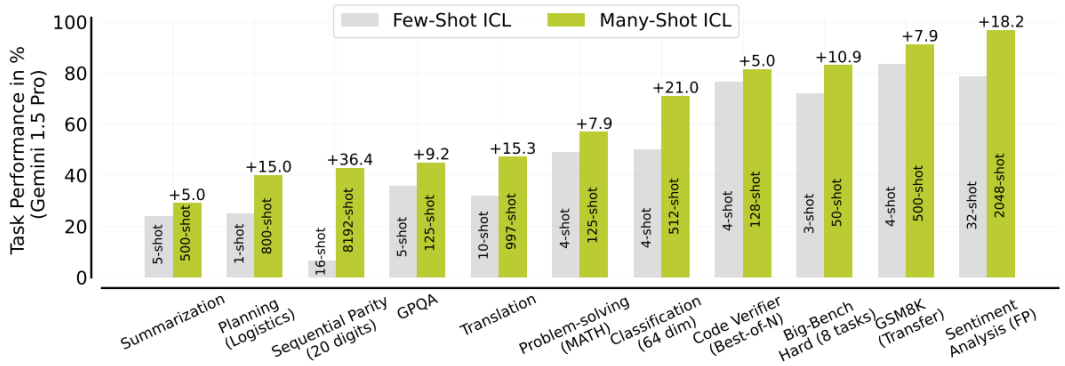

Figure 3.4: Comparison between few-shot and many-shot prompting (Agarwal et al., 2024)

Based on these findings, many-shot learning was adopted as the second component of the prompt in this project by providing the LLM with examples of NLQ-SQL pairs from the training dataset. As discussed in Section 3.2, the training set contained 51 NLQ-SQL pairs along with additional metadata. During development, a 50-50 split was used, with 25 examples for training (many-shot learning) and 26 for testing. However, once the prompt was fully engineered and finalised, all 51 NLQ-SQL pairs were utilised for many-shot learning. In addition to the NLQ-SQL pairs, I also incorporated the associated metadata for each pair to further refine the model's performance. Section 3.2 provides an overview of the different types of metadata available for each NLQ-SQL pair in the dataset. The specific components selected for the many-shot learning phase included:

- NLQ
- Notes for Student
- SQL
- 2nd SQL
- Minimum Output Columns

To integrate these components into the prompt, all 51 NLQ-SQL pairs, along with their metadata, were formatted into a structured text representation. This ensured that the LLM could learn from a diverse set of examples while considering contextual information. The prompt was designed as follows:

> **Here are some examples of natural language queries and their corresponding PostgreSQL queries:**
> - **Q:** *Natural Language Query*
> - **Notes:** *Notes for Student*
> - **A:** *SQL*
> - **Alternative A:** *2nd SQL*
> - **Min Required Columns:** *Minimum output columns*
>
> ...(for all 51 NLQ-SQL pairs)

This structured format was consistently applied across all 51 examples, serving as the foundation for the many-shot learning phase. However, this component accounted for the highest token usage, totalling **4,734 tokens**, making it the most computationally demanding and cost-intensive part of the prompt. Each user query (i.e., API call) incurred a cost of **$0.0237**.

3.3.3 Refinement Rules

The third component of the prompt incorporated refinement rules, forming a crucial part of the iterative refinement process, as shown in Figure 2.2. This refinement was performed manually on the development set, using a 50-50 split, where 25 NLQ-SQL pairs were used for training and the remaining 26 for testing. Through this process, I systematically analysed the LLM's outputs, identifying and addressing errors such as syntactic issues or overly restrictive behaviour in SQL WHERE clauses. By pinpointing areas where the LLM struggled, I iteratively developed and modified a set of refinement rules. Figure 3.5 illustrates this manual iterative refinement process.

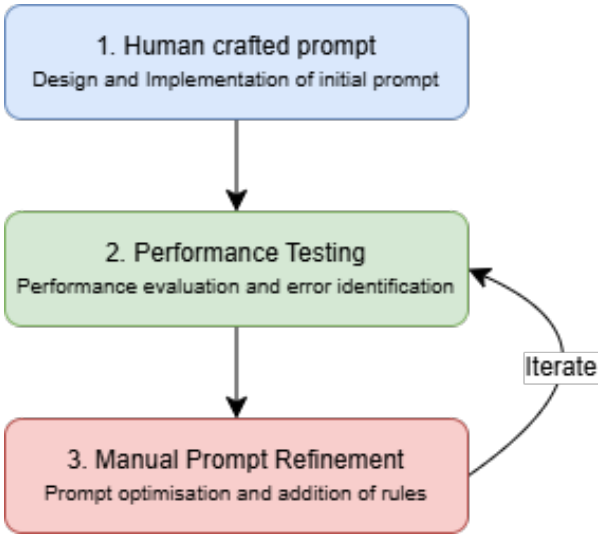

Figure 3.5: Iteration and Prompt Refinement Process

Here is how the refinement component of the prompt was designed.

Follow these strict rules:

1. Always define 'WITH RECURSIVE' at the beginning of the query if needed.
2. Ensure all subqueries are correctly nested and avoid placing 'WITH RECURSIVE' inside 'IN (...)' clauses.
3. Always use 'EXPLAIN' to verify the query structure before finalizing the query.
4. Avoid unnecessary parentheses that can cause syntax errors.
5. Make sure column names match the provided schema exactly.
6. When using 'ILIKE', ensure it applies to a valid text column and make the match **more flexible**:
   - If filtering for a **single word** (e.g., 'delta'), use:

```
ILIKE '%delta%' OR ILIKE '%Delta%'
```

   - If filtering for **multi-word terms** (e.g., 'GABA B1' or '*GABA* < *sub*> *B*1 < /*sub* >'), try **multiple formats**:

```
ILIKE '%GABAB1%' OR ILIKE '%GABA␣B1%' OR
   ILIKE '%GABA<sub>B1</sub>%'
```

7. Use table aliases to improve query readability and ensure correct joins.
8. When instructed to generate a write-up or essay, focus solely on generating PostgreSQL queries that retrieve as much relevant data as possible.

Your task is to generate valid, executable PostgreSQL queries based on natural language questions. Generate only fully functional SQL queries without placeholders or missing clauses

The following set of rules was incorporated into the prompt, resulting in optimal LLM performance during the development testing phase. Additionally, this component of the prompt is the most efficient in terms of token usage and cost, requiring **179 tokens**, which corresponds to a cost of **$0.0009** per user query (i.e., per API call).

3.3.4 Self-Correction

The final component of the prompt design incorporates self-correction, enabling an automated iterative refinement process. If an SQL query execution fails, the prompt is dynamically adjusted and retried. Figure 3.6 illustrates this process step by step. The user submits a natural language query (NLQ), which is fed into the LLM. The LLM then converts the NLQ into an SQL query, which the backend executes on the GtoPdb. We are currently focusing on the scenario where an error occurs. If the generated SQL query fails, the database returns a descriptive error message explaining the issue. The backend captures this error message and sends it back to the LLM, linking it to the original NLQ. The LLM, now incorporating the error context, generates a revised SQL query intended

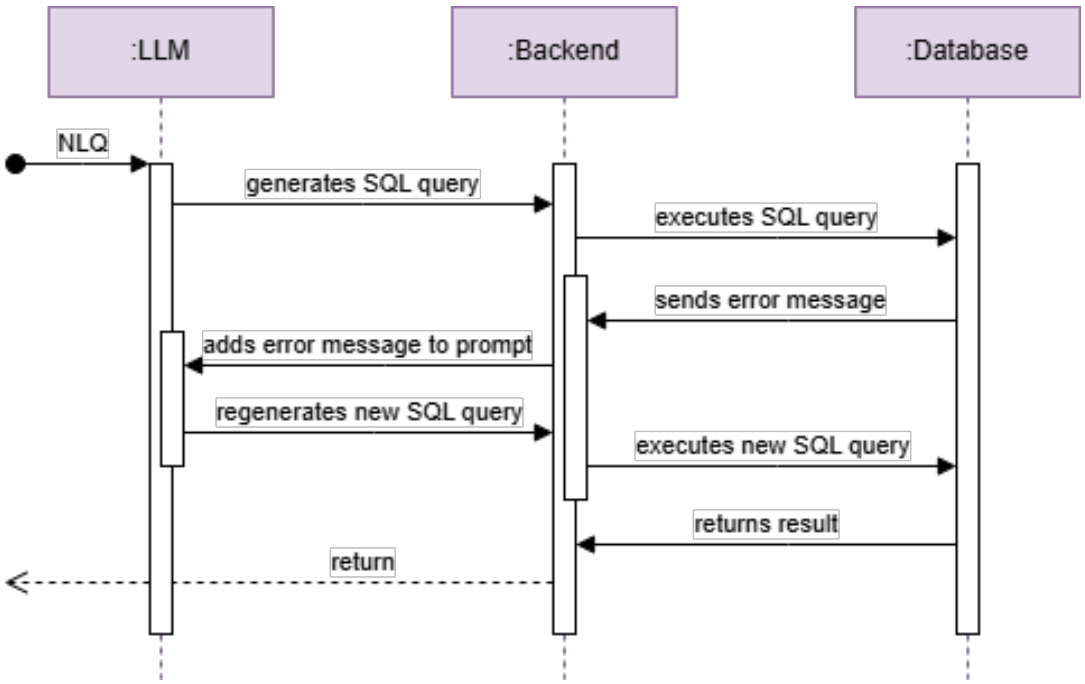

Figure 3.6: Self-Correction Process

to correct the previous mistake. This new SQL query is then executed on the database. Regardless of whether it fails again or executes successfully, the backend captures the result and presents it to the user. During the execution of an LLM-generated SQL query, two types of failures can occur, both of which must be appropriately handled to facilitate self-correction:

1. **Execution Failure** – The SQL query cannot be executed due to syntax errors or references to non-existent tables or columns. This type of SQL execution failures return self-descriptive error messages, which will be directly incorporated into the prompt. For instance, a typical SQL execution failure would look like this which is attached to the user prompt on retrying:

   Execution failed on SQL:
   ```
   WITH RECURSIVE subfamilies AS (...)
   SELECT ...
   FROM object o
   JOIN receptor2family rf ON o.object_id = rf.object_id
   ...
   WHERE name ILIKE '%chemokine receptors%'
   AND i.selective = 'Selective';
                     ^
   ```
   Error: Invalid input syntax for type boolean: "Selective"

2. **Empty Result Set** – The query is syntactically correct and executes successfully but returns no results (i.e., no rows are retrieved). Since no error message is generated by default when this occurs, a custom error message was designed to

be automatically generated and fed into the prompt.

> SQL executed but returned an empty result.
> Query: [Previously failed SQL query]
> Possible reasons:
> - The referenced table or columns may be empty.
> - Filtering conditions might be too strict.
> - Data may not exist for the given WHERE clause.
> - Joins might be eliminating rows due to unmatched conditions.

The set of reasons of failure was curated using the GPT-4o LLM, through an analysis of its output against the expected results on the development set. For both failure types, the failure message was attached to the re-attempted NLQ, ensuring that the LLM received crucial feedback to iteratively refine its SQL generation process.

Self-Correction prompt = Previous error message + NLQ

Only one round of self-correction was permitted for an NLQ–if the query fails again, it is classified as a complete failure. This approach optimises token usage while also allowing the user to modify their NLQ if necessary to obtain better results. Through this process, an optimised prompt was developed, integrating four key components–GtoPdb schema, many-shot prompting, refinement rules, and self-correction into the LLM's context for each API call, ensuring effective SQL generation for every user-provided natural language query.

3.4 Testing

A range of metrics was used to evaluate the performance of the prompt-engineered LLM for Text-to-SQL during both the development and testing phases. The test suite was carefully designed to capture all details related to SQL execution, ensuring a comprehensive assessment of performance. Four evaluation metrics were selected for assessing the LLM, determined collaboratively by myself, Ian Little–who worked on a different methodology within this project, and our supervisors, Dr. David Sterratt and Dr. Simon Harding from the IUPHAR Executive Committee.

3.4.1 Metrics

The Successful Execution Rate (SER) and Non-Empty Execution Rate (NER) measure the percentage of generated SQL queries that are syntactically correct, executable, and capable of returning results. Meanwhile, Execution Accuracy (EX) and Partial Execution Accuracy (PEX) compare the actual output of the SQL query against the expected data, quantifying the similarity between them. For both the development and test sets, each metric was evaluated as a boolean value, indicating whether it was satisfied for a given query. The final metric score was then computed as the percentage of queries for which the metric was marked as true. Let M be a given metric (e.g., SER,

NER, EX, or PEX), and let there be N total queries in the dataset (either development or test set). Define an indicator function:

$$I_M(q_i) = \begin{cases} 1, & \text{if the metric } M \text{ is satisfied for query } q_i \\ 0, & \text{otherwise} \end{cases}$$

Then, the final metric score for M is computed as:

$$\text{Final Score}_M = \frac{1}{N} \sum_{i=1}^{N} I_M(q_i) \times 100\%$$

This formula expresses that for each query q_i, the metric is evaluated as a boolean value (1 if satisfied, 0 otherwise), and the final percentage score is the average number of queries for which the metric was satisfied. The goal through the development phase was to achieve a value has high as possible on all four of the following metrics.

Successful Execution Rate

This metric quantifies the percentage of test queries that executed successfully, meaning they were syntactically valid and executed on the database without errors. Notably, if an SQL query runs but returns an empty table (i.e., zero rows), it is still considered a successful execution.

Non-Empty Execution Rate

This metric quantifies the percentage of test queries that executed successfully and also returned non-empty tables. Therefore, if an SQL query runs but returns an empty table, the query would be considered as a failed execution.

Execution Accuracy

This metric evaluates the correctness of SQL query execution by comparing the results of the generated SQL query against a gold standard SQL query derived from the NLQ-SQL pairs in the dataset. Both the generated and gold standard SQL queries are executed, and the generated query is considered correct for this metric only if all returned rows match exactly. However, actual column names are not taken into account.

Let:

- P be the set of rows returned by the **predicted SQL query** (i.e., `predicted dataframe`).
- G be the set of rows returned by the **gold standard SQL query** (i.e., `gold dataframe`).

The EX function per query is defined as:

$$\text{Query EX}(P,G) = \begin{cases} 1, & \text{if } P = G \\ 0, & \text{otherwise} \end{cases}$$

where:

- Each row in P and G is treated as an unordered tuple, meaning row and column order does not affect the comparison.
- If either P or G is `None` (i.e., the query did not return results), the function returns 0.

Partial Execution Accuracy

This metric assesses the correctness of SQL query execution by comparing the results of the generated SQL query with a gold standard SQL query derived from the NLQ-SQL pairs in the dataset. Both queries are executed, and the generated query is considered correct under this metric only if all returned rows match, while variations in the number of columns are permitted. Essentially, this ensures that the same data is retrieved, regardless of whether additional or fewer columns are present. If common columns exist, only the data from those columns are compared between the results of both queries. If no common columns exist, the evaluation follows the same logic as EX. The PEX function can be formulated mathematically as follows.

Let:

- P be the set of predicted rows from the generated SQL query.
- G be the set of gold standard rows from the reference SQL query.
- C_P be the set of columns in the predicted query's result.
- C_G be the set of columns in the gold standard query's result.
- P' and G' be the filtered versions of P and G, containing only columns in $C_P \cap C_G$.

PEX function per query is defined as:

$$\text{Query PEX} = \begin{cases} 1, & \text{if } C_P \cap C_G \neq \emptyset \text{ and } P' = G' \\ 1, & \text{if } C_P \cap C_G = \emptyset \text{ and } \mathit{Query\ EX}(P,G) = 1 \\ 0, & \text{otherwise} \end{cases}$$

Each of these four metrics was averaged across all test queries, and a final percentage was computed for each metric.

3.4.2 Process

The testing process was conducted in two phases, development testing, which occurred during development, and final testing, which took place after development was completed.

Development Testing

During the development phase, only the development set of NLQ-SQL pairs was used for evaluation, as outlined in Section 3.2. The development set consisted of 51 NLQ-

SQL pairs, evenly distributed across four difficulty levels. Of these, 26 queries were selected for testing the prompt, with a 50-50 split, and all four metrics were applied to evaluate performance.

Final Testing

After finalising the prompt, the version that demonstrated the best performance across the four metrics during development testing was selected. As mentioned in Section 3.2, the specially designated test set consisted of 30 NLQ-SQL pairs. This test set was only revealed once development was complete to ensure an unbiased development and assessment process. Final testing was performed on this set of pairs, with all four metrics calculated to assess the prompt's performance on an unseen dataset. The results were also compared with Ian Little's solution for the same task, which will be discussed in depth in Chapter 4.

Chapter 4

Results and Discussion

4.1 Previous Experiments

The approach taken in this study was to provide the LLM with relevant information in a structured and concise manner, optimising its performance for the given task. To assess its incremental improvements, information was introduced step by step. Evaluation began with Non-Empty Output Rate (NER) as the primary metric to systematically refine the prompt's components.

These evaluations were conducted on a 50% split of the development set, comprising 26 out of 51 NLQ-SQL pairs. As illustrated in Figure 4.1, providing only the manual refinement rules resulted in the lowest NER score of 0%. In the Many-Shot Learning (MSL) setting, a 25-shot prompt without additional context achieved approximately 30% NER. Subsection 4.1.1 highlights how the optimal n was found for this component. The most significant improvement occurred when the schema structure was included, increasing NER to nearly 40%. Alternative methods were explored for incorporating schema information, such as mapping all possible columns to their respective tables instead of tables to columns to reduce token usage. However, this did not significantly decrease the number of tokens. Another approach, using numerical IDs to reference columns, was discarded as the LLM requires interpretable natural language prompts.

To further enhance performance, all three components–rules, MSL, and schema were integrated into a single prompt. This resulted in a sharp increase in NER to approximately 70%. Finally, self-correction (SC) was employed for error handling, allowing the LLM to refine its SQL generation after receiving error messages from previous attempts. This approach led to a substantial performance gain, with NER exceeding 90%. These findings demonstrated a clear trend: performance improves as rules, MSL, and schema information are introduced, with a notable boost when combined. Self-correction further enhances accuracy, underscoring its potential for iterative SQL generation refinement.

For the visualisation of results, error bars represent 95% confidence intervals for proportions, computed using the Jeffreys interval[1] , a Bayesian approach based on

[1]Confidence Interval: `https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval`

Jeffreys prior (Beta(0.5, 0.5)). Given a sample size n and observed proportion p, the Beta distribution parameters are set as:

$$a = n \cdot p + 0.5, \quad b = n \cdot (1 - p) + 0.5$$

The lower and upper bounds are then obtained using the inverse survival function (ISF) of the Beta distribution at the 2.5% and 97.5% percentiles, providing a credible range for the true proportion. This approach highlights the variance in the LLM's behaviour when generating outputs, indicating the expected variation if the experiment were repeated multiple times.

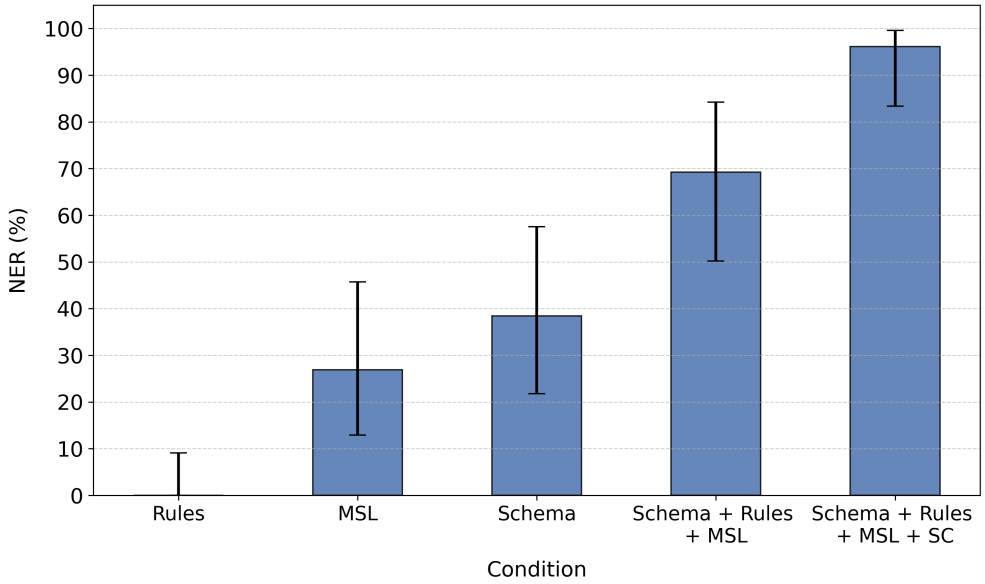

Figure 4.1: Effect of Incrementally Adding Information on LLM Performance for NER

4.1.1 Finding the optimal *n* for *n*-Shot Learning

To implement Many-Shot Learning effectively, it was necessary to determine the optimal number of NLQ-SQL examples (n) to include. Experiments were conducted from 0-shot up to 25-shot learning, while keeping all other prompt components constant. n samples were chosen at random from the 50% split on the development set while testing on the remaining 50%. Figure 4.2 illustrates the impact of increasing n on the NER, Partial Execution Accuracy (PEX) and Execution Accuracy (EX) metrics. Additionally, Appendix B presents the effect on Successful Execution Rate (SER) though this metric exhibited negligible variation. For NER, a steady increase was observed, starting at over 50% for 0-shot and exceeding 95% at 25-shot. A notable spike occurred at 5-shot, reaching 80%, though the improvement was not as pronounced as at 25-shot. PEX showed a more gradual increase, remaining consistently below 25% between 5-shot and 15-shot, before rising to approximately 30% at 20-shot and 25-shot. In contrast, EX began at around 7% for 0-shot, followed by a slight decrease from 1-shot to 5-shot, remaining under 5%. However, from 10-shot to 20-shot, EX remained above 10%, with a final increase at 25-shot, where EX exceeded 15%.

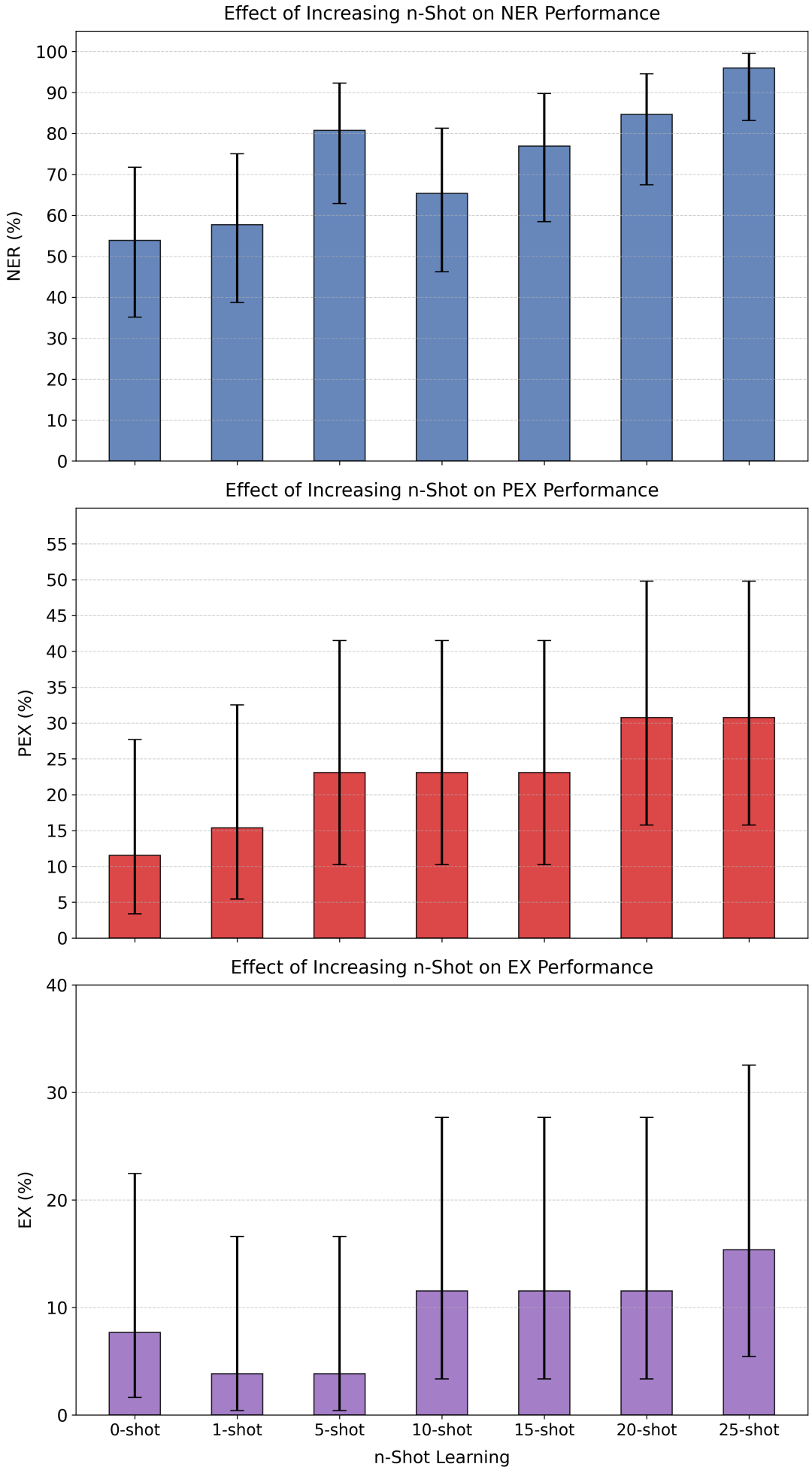

Figure 4.2: Effect of increasing n-shot in NER, PEX and EX performance

4.1.2 Ablation study on Individual Prompt Components

An ablation experiment was designed to evaluate the significance of each individual component within the prompt. Ablation methods are employed in prompt engineering and LLM research to assess the impact of specific prompt components on performance (Maharjan et al., 2024; Zhang et al., 2024). Given that the prompt consisted of four distinct components, a series of ablation experiments were conducted on the development set. Each component was systematically removed from the full prompt, and the resulting performance was compared against that of the complete prompt. Figures Figure 4.3 illustrates the impact of excluding schema structure, many-shot learning, rules, and self-correction.

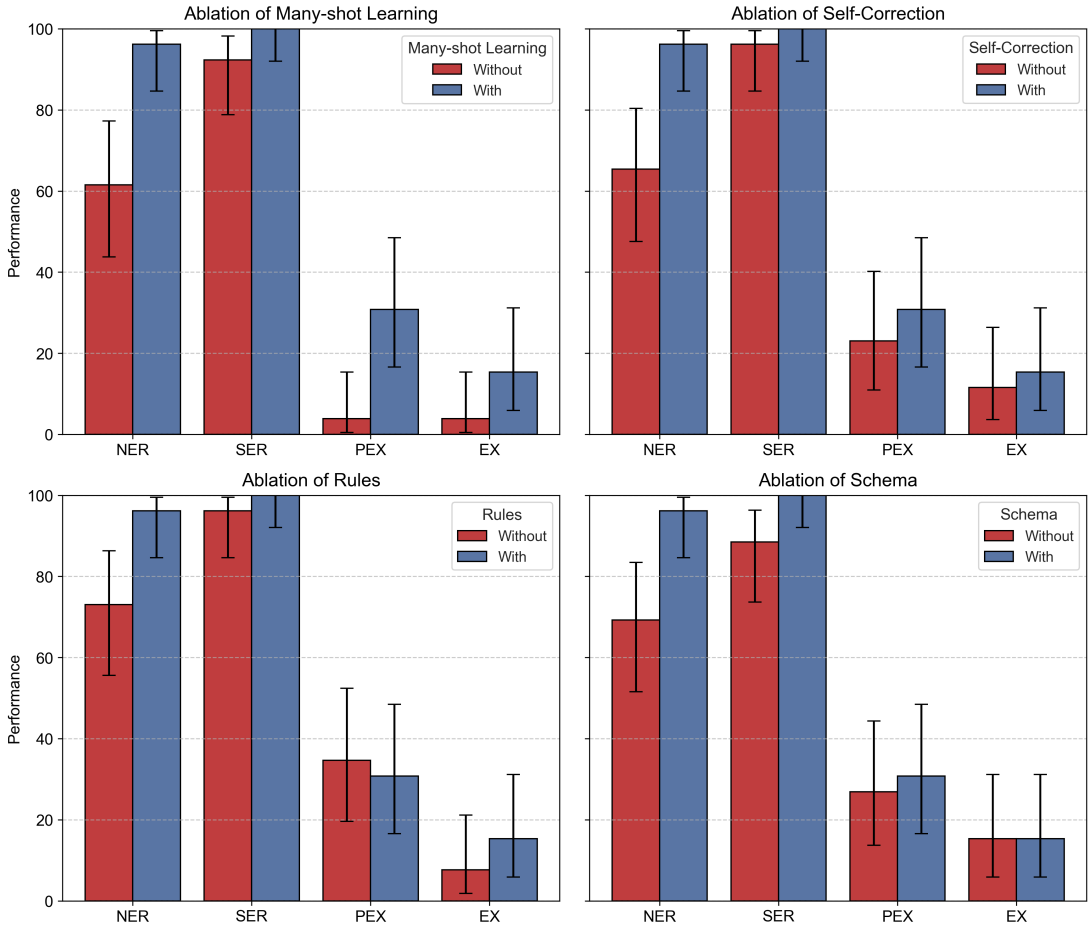

Figure 4.3: Effect of prompt components in LLM performance

Schema

The removal of the schema had a significant impact, particularly on the NER and SER metrics, while its effect on PEX and EX was minimal. This suggests that the schema plays a crucial role in ensuring the generated SQL query is syntactically correct and executable, rather than directly influencing whether the query retrieves the intended results. Specifically, excluding the schema led to a 25% decrease in NER and a 10% reduction in SER.

Many-shot Learning

The removal of Many-Shot Learning resulted in a substantial decline in performance, with NER decreasing by approximately 35%. This also led to the most significant drops in PEX and EX, with reductions of 30% and 10%, respectively. Notably, without Many-Shot Learning, PEX and EX were close to 0%, indicating that providing relevant NLQ-SQL examples is crucial not only for generating syntactically correct and executable SQL queries (as reflected by NER) but also for ensuring that the queries retrieve the intended results, as measured by PEX and EX.

Rules

One of the queries, "Write an essay on endothelin receptor antagonists," timed out as it failed to generate an SQL query when the rules were removed. The absence of rules also led to a more than 20% decrease in NER, indicating their crucial role in ensuring that the generated SQL queries are valid, executable, and non-empty. While PEX appeared higher, this is likely due to variance and inconsistencies in LLM behaviour, as reflected by the error bars. Additionally, EX dropped significantly by 10%, further highlighting the importance of rules in query formulation.

Self-Correction

Lastly, the removal of self-correction resulted in a notable decrease of over 30% in NER. PEX also dropped by approximately 10%, while SER and EX showed reductions, though these were not as substantial. This suggests that self-correction plays a key role in refining query generation, particularly in improving syntactic correctness and execution accuracy.

It is evident that all four components of the prompt play a crucial role in maximising NER, SER, PEX, and EX. The removal of any component resulted in a decline in the LLM's performance. Additionally, the experiments demonstrated that increasing n in n-shot learning significantly enhances performance, with Many-Shot Learning having the most substantial impact on ensuring high PEX and EX scores.

4.2 Final Approach

Based on these experiments, it was determined that all four prompt components should be retained, and n-shot learning should be maximised by utilising all available examples. Since the development set contained 51 NLQ-SQL pairs, 26 were used for Many-Shot Learning, while the remaining 25 were reserved for testing. However, for the test set, which consisted of 30 NLQ-SQL pairs designated solely for testing, a 51-shot learning approach was implemented using the entire training set.

4.2.1 Development Set Results

The results from the development set in Table 4.1 demonstrate the effectiveness of the final prompt in generating executable SQL queries. The LLM achieved an SER of 100%

on 26 test queries all together. Notably, 96.15% of these queries produced non-empty outputs, suggesting that the prompt effectively guides the LLM to generate meaningful queries that retrieve relevant data. However, the EX is at 15.38%, indicating that there are areas for future work in generating SQL that precisely aligns with user intent on the LLM's first attempt itself. PEX is recorded at 30.77%, which is double as EX, meaning that the LLM does capture the data the user intends to see but with more or less columns. Appendix C depicts these results in further detail.

4.2.2 Test Set Results

The test set, consisting of 30 NLQ-SQL pairs, was released after the development and finalisation of the prompt. As mentioned in Table 4.1, the SER remained consistent with the development set results, achieving 100%. However, the NER was slightly lower at 90%, compared to the development set. EX also saw a minor decline, with a value of 13.33%, about 2% lower than in the development set. On the other hand, PEX increased, reaching 43.33%, up from 30.77% in the development set. Appendix D depicts these results in further detail. The results of the ablation study on Many-Shot Learning indicated that it played a critical role in enhancing the PEX and EX values, as shown in Figure 4.3. This increase could be attributed to the shift from 25-shot learning (used during development testing) to 51-shot learning, which incorporated the entire training set and likely improved the PEX. Additionally, the upward trend in PEX with increasing n in n-shot learning, as depicted in Figure 4.2, further supports this explanation.

Metric	**Development Set**		**Test Set**	
	Value	**95% CI**	**Value**	**95% CI**
Count	26	-	30	-
SER	100.00%	90.88% - 100.00%	100.00%	92.03% - 100.00%
NER	96.15%	83.39% - 99.58%	90.00%	75.66% - 97.10%
PEX	30.77%	15.75% - 49.81%	43.33%	26.89% - 60.99%
EX	15.38%	5.42% - 32.54%	13.33%	4.67% - 28.65%

Table 4.1: Comparison of Development and Test Set Results

4.3 Comparison with Alternative Prompt Implementation Strategy

In the companion undergraduate project that Ian Little implemented, an alternative prompt engineering strategy for text-to-SQL on GtoPdb was implemented using the o3-mini model, as illustrated in Figure 4.4. This approach follows a multi-layered LLM querying process that involves the following stages:

1. Schema Linking: Given an NLQ, the LLM is provided with the full schema and instructed to select the most relevant tables.

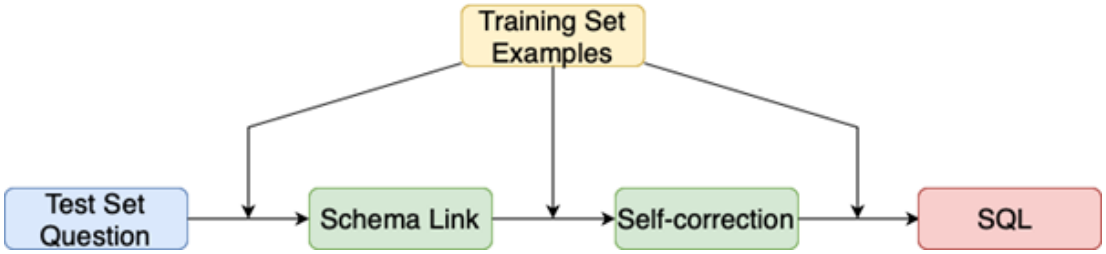

Figure 4.4: Alternative Prompt Implementation Strategy (Little, 2025)

2. SQL Generation: Using these selected tables as context, the LLM generates an SQL query.
3. Self-Correction (Optional): If an error is detected, the LLM regenerates an SQL query by sending the error message as additional context.

A 5-shot learning approach is applied between these stages, where the LLM selects five NLQ-SQL examples based on their semantic relevance to the user's NLQ rather than through random sampling. These examples are incorporated into the prompt at each stage of the LLM querying process. Across the pipeline, for each user query, the LLM is queried up to four times–once for retrieving the top 5 NLQ-SQL training set samples, once for schema linking, once for SQL generation, and optionally for self-correction.

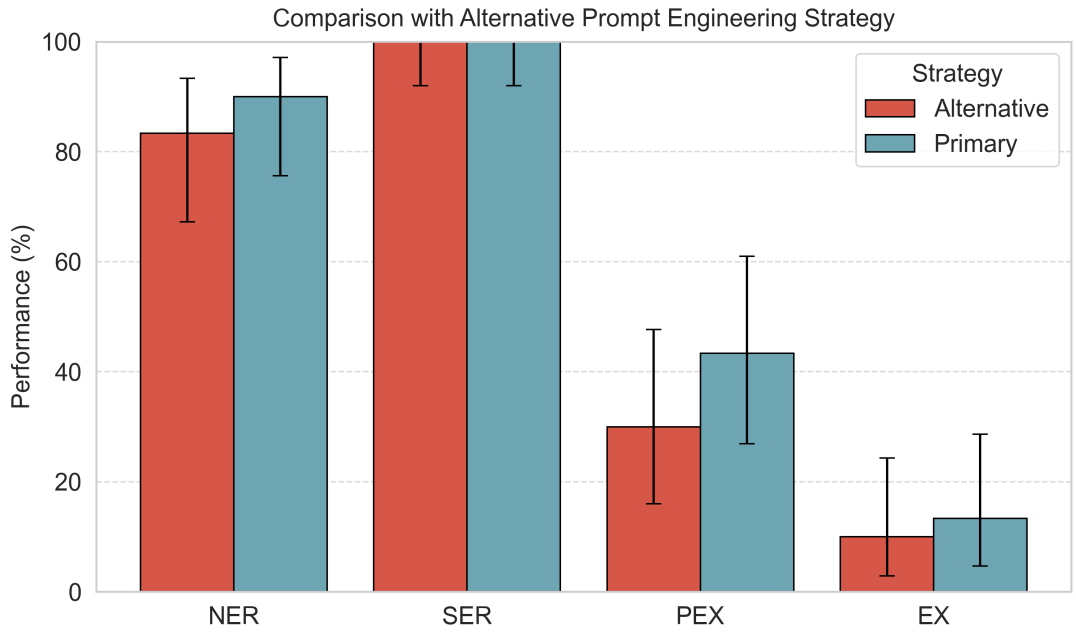

Figure 4.5: Comparison between the primary approach (this project) and the alternative prompt implementation strategy by Ian Little

Figure 4.5 presents a comparative analysis of NER, SER, PEX, and EX between the primary approach (this project) and the alternative method implemented by Ian Little (Little, 2025). Both models were evaluated using the same test set of 30 queries. SER was identical for both at 100%, and the NER difference was 6.67%, indicating that both

approaches generate valid and executable queries. A 3.33% difference was observed in EX, showing a variation in the accuracy of retrieving the expected results. The most notable contrast was in PEX, where the primary approach outperformed the alternative by 13.33%. This is likely due to the difference in the number of examples provided during training—51-shot learning in this project versus 5-shot learning in the alternative approach. The results suggest that while schema linking helps narrow down relevant tables, providing a larger set of examples significantly impacts PEX to understand user needs from a single prompt better.

4.4 Runtime and Cost

The average runtime per query on the final test set was 6.01 ± 5.14 seconds, where 5.14 is the standard deviation [2]. This was computed by summing the runtimes of each NLQ-SQL in the test set (size = 30) and calculating the mean and standard deviation. The cost breakdown for the three compulsory components of the prompt is depicted in Table 4.2.

Prompt Component	Tokens	Cost ($)
Database Schema	2495	0.0125
Many-shot Prompting	4734	0.0237
Refinement Rules	179	0.0009
Total	**7408**	**0.0370**

Table 4.2: GtoPdb LLM System Prompt's Token usage and cost

In the event that the initial LLM-generated SQL query fails, self-correction is employed to generate a revised query. This process typically leads to a doubling of the token usage from 7408 tokens to approximately 14816 tokens. As a result, the total cost per query would increase, with the cost rising to approximately $0.0741 when self-correction is triggered.

As the cost of providing context to GPT-4o is $5 per million tokens, based on the current prompt engineering setup, the minimum cost per API call is $0.0370. For each user query, the cost will either be $0.0370, assuming no failure and self-correction, or approximately $0.0741 if self-correction is required due to a failed initial query. This calculation considers only the cost of prompt engineering, excluding additional tokens required for output generation and query processing, which are likely minimal.

[2]Standard Deviation: `https://en.wikipedia.org/wiki/Standard_deviation`

Chapter 5

Conclusions

5.1 Summary and Reflection

A prompt was engineered for the GPT-4o model to convert natural language queries (NLQs) into SQL for the Guide to Pharmacology database (GtoPdb). This prompt consisted of four key components: the GtoPdb schema, many-shot learning with 51 NLQ-SQL pairs from the training set, manually written rules, and optional self-correction for error correction. A dataset curated by the NC-IUPHAR Database Executive Committee was used, containing 51 NLQ-SQL pairs for the training set and 30 NLQ-SQL pairs for the test set, along with additional metadata. To evaluate the performance of the LLM, four key metrics were used: Successful Execution Rate (SER), Non-Empty Output Rate (NER), Partial Execution Accuracy (PEX), and Execution Accuracy (EX). SER and NER measure whether the LLM generates syntactically correct and non-empty outputs, while EX assesses whether the SQL results precisely match user intent. PEX evaluates the same, but with variations in the number of columns.

During development, a 50-50 split was used, with half the dataset allocated for testing and the remainder for many-shot learning. It was demonstrated that increasing the number of NLQ-SQL pairs in many-shot learning improved performance in NER, PEX and EX, indicating that all available pairs should be utilised. An ablation study further confirmed the importance of each prompt component, as removing any of the four elements led to a decline in performance across all metrics.

The final prompt was tested on the 30-query test set, achieving SER = 100%, NER = 90%, PEX = 43.33%, and EX = 13.33%. These results were compared with an alternative approach implemented by a fellow student, which used schema-linking and 5-shot learning (selecting the most relevant examples for each NLQ). Notable differences were observed in PEX, where the alternative method performed worse, likely due to the difference between 5-shot and the 51-shot strategy used in this study. SER remained the same at 100%, with minor performance variations in NER and EX. The runtime of the final prompt on the test set averaged 6.01 ± 5.14 seconds per user query (where 5.14 is the standard deviation), with a token usage of 7408 and a cost of $0.0370 per user query. If self-correction were applied, this cost would approximately double.

Compared to the BIRD benchmark, where the best model achieved an EX of 54.89% (with human performance at 92.96%) (Li et al., 2023), the GtoPdb LLM attained an EX of only 13.33%. A key factor in this performance gap is likely the difference in training data: the BIRD training set comprises 11,218 NLQ-SQL pairs across 95 diverse databases, whereas the GtoPdb LLM relied on only 51 examples for many-shot learning. This stark contrast in data volume–51 samples versus 11,218–could explain the disparity in execution accuracy. Notably, GPT-4 achieved 54.89% EX on BIRD, and this project used an even more advanced model, GPT-4o, with access to all relevant knowledge. Given this, performance should theoretically be comparable. However, the complexity of the GtoPdb itself, adds an additional challenge beyond data sparsity. Expanding the training dataset is expected to improve the EX of the GtoPdb LLM, but due to the complexity of the database, it is less likely to reach the EX seen in the BIRD benchmark.

5.2 Areas of Future Work

Since NER and SER demonstrated high results, the next priority is enhancing PEX and EX, which were recorded at 43.33% and 13.33% respectively, to ensure that the generated SQL queries precisely match user intent. Many-shot learning emerged as an influential factor in enhancing EX and PEX, suggesting that providing a larger set of example NLQ-SQL pairs could significantly improve performance. The BIRD benchmark further supports this, demonstrating that more examples correlate with higher EX scores.

Another potential direction to refine PEX and EX is the incorporation of user feedback into the query generation process. Developing a user interface (UI) would enable users to interact with and refine the generated SQL queries, such as by requesting more or fewer columns or directly editing the query. Human intervention can enhance text-to-SQL accuracy (Cai et al., 2024). The UI could also leverage contextual chains, such as LangChains [1], to retain the context of previous queries once a session starts, allowing the LLM to perform more effectively while optimising token usage (Wu et al., 2021). Additionally, integrating Retrieval-Augmented Generation (RAG), where specific tools and predefined functions supplement the LLM's capabilities, would reduce reliance on the model alone. This approach would provide greater control over the query generation process and enable more precise tuning to meet user-specific needs (Cuconasu et al., 2024).

Further optimisation efforts should focus on reducing token usage, computational costs, and query efficiency. Techniques similar to schema-linking and 5-shot learning-where the most relevant examples are dynamically selected based on the NLQ-could be explored to optimise performance while maintaining accuracy. These improvements, combined with a more interactive and adaptive user experience, would help address the current limitations and enhance the overall effectiveness of text-to-SQL conversion for the GtoPdb.

[1] LangChain: `https://www.langchain.com/`

Bibliography

Agarwal, R., Singh, A., Zhang, L. M., Bohnet, B., Rosias, L., Chan, S. C., Zhang, B., Faust, A., and Larochelle, H. (2024). Many-shot in-context learning. In *ICML 2024 Workshop on In-Context Learning*. Last Modified: 02 Jul 2024.

Cai, Y., Mao, S., Wu, W., Wang, Z., Liang, Y., Ge, T., Wu, C., WangYou, W., Song, T., Xia, Y., Duan, N., and Wei, F. (2024). Low-code LLM: Graphical user interface over large language models. In Chang, K.-W., Lee, A., and Rajani, N., editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations)*, pages 12–25, Mexico City, Mexico. Association for Computational Linguistics.

Chang, S. and Fosler-Lussier, E. (2023). How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. NeurIPS 2023 Table Representation Learning Workshop.

Cuconasu, F., Trappolini, G., Siciliano, F., Filice, S., Campagnano, C., Maarek, Y., Tonellotto, N., and Silvestri, F. (2024). The power of noise: Redefining retrieval for rag systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24)*, pages 719–729. ACM.

Drushchak, N., Polyakovska, N., and Zikrach, D. (2024). Master the art of prompt engineering: Strategies for optimization and evaluation. White Paper. Accessed: 2025-03-22.

Harding, S. D., Armstrong, J. F., Faccenda, E., Southan, C., Alexander, S. P. H., Davenport, A. P., Spedding, M., and Davies, J. A. (2024). The iuphar/bps guide to pharmacology in 2024. *Nucleic Acids Research*, 52(D1):D1438–D1449.

Hong, Z., Yuan, Z., Zhang, Q., Chen, H., Dong, J., Huang, F., and Huang, X. (2024). Next-generation database interfaces: A survey of llm-based text-to-sql. arXiv preprint, version 5, last revised 13 Mar 2025.

Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Chenhao, M., Li, G., Chang, K., Huang, F., Cheng, R., and Li, Y. (2023). Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023) Datasets and Benchmarks Track*.

Little, I. (2025). Leveraging large language models for text-to-sql on the iuphar/bps guide to pharmacology database. Unpublished dissertation.

Maharjan, J., Garikipati, A., Singh, N. P., Cyrus, L., Sharma, M., Ciobanu, M., Barnes, G., Thapa, R., Mao, Q., and Das, R. (2024). Openmedlm: Prompt engineering can out-perform fine-tuning in medical question-answering with open-source large language models. *Scientific Reports*, 14.

Matarazzo, A. and Torlone, R. (2025). A survey on large language models with some insights on their capabilities and limitations. arXiv preprint, version 2, last revised 9 Feb 2025.

Meyer, J. G., Urbanowicz, R. J., Martin, P. C. N., O'Connor, K., Li, R., Peng, P.-C., Bright, T. J., Tatonetti, N., Won, K. J., Gonzalez-Hernandez, G., and Moore, J. H. (2023). Chatgpt and large language models in academia: opportunities and challenges. *BioData Mining*, 16(20).

Nan, L., Zhao, Y., Zou, W., Ri, N., Tae, J., Zhang, E., Cohan, A., and Radev, D. (2023). Enhancing text-to-sql capabilities of large language models: A study on prompt design strategies. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14935–14956, Singapore. Association for Computational Linguistics.

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., and Mian, A. (2023). A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.

Pourreza, M. and Rafiei, D. (2023). DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *arXiv preprint arXiv:2304.11015*. To appear in NeurIPS 2023.

Shen, J., Wan, C., Qiao, R., Zou, J., Xu, H., Shao, Y., Zhang, Y., Miao, W., and Pu, G. (2025). A study of in-context-learning-based text-to-sql errors.

Tabarsi, B., Reichert, H., Limke, A., Kuttal, S., and Barnes, T. (2025). Llms' re-shaping of people, processes, products, and society in software development: A comprehensive exploration with early adopters. arXiv preprint.

Truhn, D., Reis-Filho, J. S., and Kather, J. N. (2023). Large language models should be used as scientific reasoning engines, not knowledge databases. *Nature Medicine*, 29:2983–2984.

Weber, I. (2024). Large language models as software components: A taxonomy for llm-integrated applications. arXiv preprint.

Wretblad, N. and Gordh Riseby, F. (2024). Bridging Language & Data: Optimizing Text-to-SQL Generation in Large Language Models. Master's thesis, Linköping University, Department of Computer and Information Science. LIU-IDA/LITH-EX-A–24/001–SE. Supervisor: Oskar Holmström. Examiner: Marco Kuhlmann.

Wu, T., Terry, M., and Cai, C. J. (2021). Ai chains: Transparent and controllable

human-ai interaction by chaining large language model prompts. *arXiv preprint arXiv:2110.01691*.

Yildirim, I. and Paul, L. A. (2024). From task structures to world models: what do llms know? *Trends in Cognitive Sciences*, 28(5):404–415.

Zhang, F. et al. (2024). Benchmarking biomedical relation knowledge in large language models. In Peng, W., Cai, Z., and Skums, P., editors, *Bioinformatics Research and Applications. ISBRA 2024*, volume 14955 of *Lecture Notes in Computer Science*. Springer, Singapore.

Appendix A

NLQ to data Example

Below is an example of a potential user request along with its generated SQL query and result.

1. User prompts: “View All Ligands”

2. Generated SQL query by GPT-4o:

```
SELECT * FROM ligand;
```

3. Retrived data from PostgreSQL:

ligand_id	name	pubchem_sid	...	bioactivity_comments_vector
3560	gastrin-17	135651757.0	...	None
3550	CCK-33	135651649.0	...	None
5237	p122-RhoGAP	178101921.0	...	None
5007	inhibin βB	178101704.0	...	None
3738	PTHrP	135651870.0	...	None
...	...	...	...	...
12728	azidamfenicol	479821188.0	...	’activ’:12 ’antibacteri’:11 ’data’:8 ’drug’:15
12729	EN67	479821427.0	...	None
12730	sutezolid	483123320.0	...	’-0.50’:18 ’0.03’:17 ’0.125’:10 ’activ’:7,36
12731	sarecycline	485206042.0	...	’-16’:12 ’0.5’:11 ’acn’:6 ’activ’:3 ’also’:24
12732	florfenicol	485206044.0	...	’-6.3’:27 ’0.4’:26 ’0.5’:23 ’activ’:5,18 ’aure...

Appendix B

n-shot performance on SER

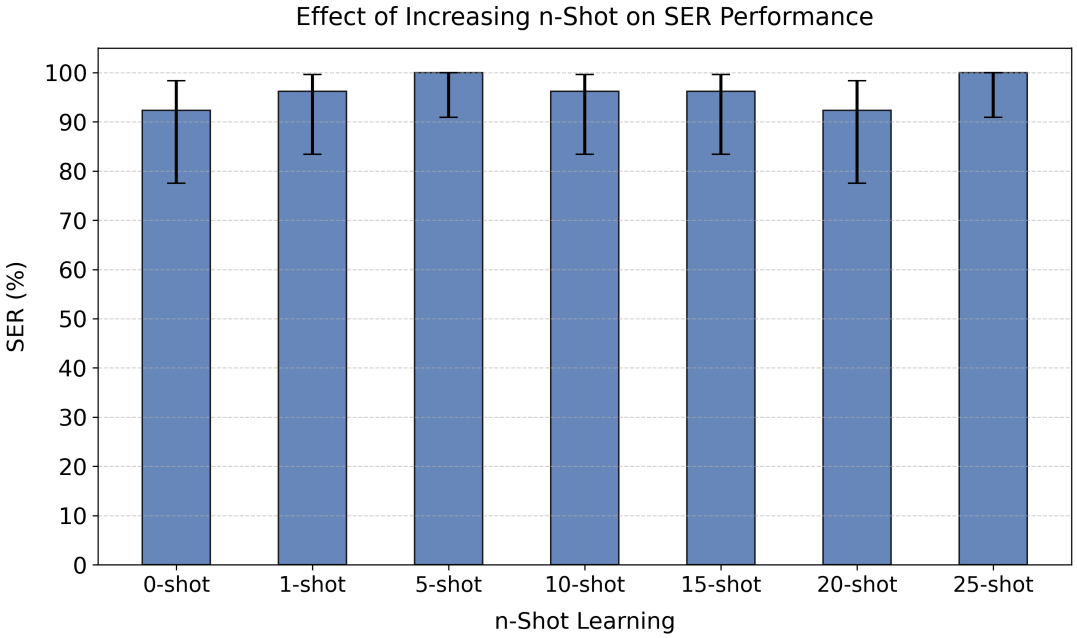

Figure B.1: Effect of increasing *n*-shot in SER performance

Appendix C

Detailed Prompt performance on development set

Table C.1: Development Set Evaluation Results

Test Case	NLQ	EX	PEX
1	What does the Guide to Pharmacology know about Voltage-gated ion channels?	False	True
2	What antimalarial ligands are approved drugs?	False	False
3	Find enzymes with endogenous substrates that have no pharmacology to speak of?	False	False
4	What ligands, not in human, interact with GABAB1	False	False
5	How many antibodies with binding data are there in GtoPdb?	True	True
6	List all fatty acid binding proteins and their human gene identifiers	False	False
7	Find synthetic organic ligands with a molecular weight less than 500	False	False
8	List chemical structures for BACE1 that have pAct $>$ 7 and order these high to low	False	False
9	Find any endogenous substrates of decarboxylases?	False	False
10	Write an essay on endothelian receptors antagonists	False	False
11	What compounds, that are not approved drugs, have evidence of being used in clinical trials?	True	True
12	Find GPCR agnonists with pKi affinities greater than 10	False	False
13	Find information on the clinical use of drugs targetting Glucagon receptors	False	False
14	Which approved antibodies target ligands?	False	True
15	What antimalarial compounds are in the database?	False	True
16	What are the selective antagonists for Leukotriene receptors and their affinities?	False	False
17	What is known about Opioid delta receptor agonists?	False	False
18	What compounds have a role in treating arthritis?	True	True

Continued on next page...

Table C.1 (Continued)

Test Case	NLQ	EX	PEX
19	Find all the papers in the Guide that are listed as preprints (useful for housekeeping)	False	False
20	Are there compounds which can block all adenosine, muscarinic, LPA, cannabinoid or dopamine receptors? Originally: Are there compounds which can block all adenosine (muscarinic, LPA, cannabinoid, dopamine) receptors?	False	False
21	What should I use to inhibit TRPM3 in cultured cells, and at what concentration?	False	False
22	Find SMILES and InChiKey for alitretinoin	True	True
23	Is there recommended background reading on endothelin receptors?	False	False
24	Find apelin ligands that are radiolabelled together with their pKDs.	False	True
25	Find natural products ligands, that meet lipinski rule-of-5 and have SMILES	False	False
26	What are the affinity values for SLC29 family inhibitors?	False	False

Appendix D

Detailed Prompt performance on test set

Table D.1: Test Set Evaluation Results

Test Case	NLQ	EX	PEX
1	What pharmacology data is there for UniProtKB ID Q8NEC5 and Q96P56?	False	True
2	What selective agonists are there for chemokine receptors family members?	False	False
3	Which Lipinski compounds target Coronavirus proteins?	False	True
4	Find all of the pharmacological agents that are active in humans but not rats and mice	False	False
5	Find all patent references in the Guide that are post-2021?	False	True
6	What fraction of all targets with clinically approved drugs are targeted by a drug on the WHO essential medicines list?	False	False
7	Which compounds have a role in acute lymphocytic leukemia?	True	True
8	What peptides, that have pharmacology in humans, also have amino-acid sequence data in the Guide?	False	False
9	What should I use to inhibit TLR7 in cultured cells, and at what concentration?	True	True
10	What antibacterial compounds are in the database?	False	True
11	Which antibodies have clinical trials data in GtoPdb?	False	True
12	List chemical structures for SARS Cov2 MPro that have pAct ¿ 7 and order these high to low	False	False
13	Find all immune checkpoint ligands	False	False
14	Which ligands have interaction data in the plasmodium asexual blood stage	True	True
15	Find NHRs that have quantitative interaction data?	True	True
16	Which compounds act as both agonists and antagonists against GPCRs?	False	False

Continued on next page...

Table D.1 (Continued)

Test Case	NLQ	EX	PEX
17	Which channel blockers act against Ryanodine receptors?	False	True
18	Find pKi value, above 6, for Ion Channel approved drugs	False	False
19	Find drugs for which calcitonin receptors are a primary target?	False	False
20	Which compounds, that also have ChEMBL Ligand IDs, target dopamine receptors?	False	True
21	Are there any approved drugs with unclear bioactivity or molecular mechanisms of action?	False	True
22	What potassium channels does GtoPdb have data on?	False	True
23	What primary references are used to curate Somatostatin receptor agonists?	False	False
24	What are the SMILES and InChI Keys for compounds that interact with Nicotinic acetylcholine receptors (nACh)?	False	False
25	What ligands and structures have pKi values > 8 when interacting with transporter targets?	False	False
26	What range of affinity values does α-MSH have for MC3 receptor?	False	False
27	What is the average pKi value for natural products compared to the average pIC50 values?	False	False
28	List endogenous activators of Adenyl cyclases, with their affinities	False	False
29	What concentration/s should be used in isolated tissue experiments?	False	False
30	Are the compounds active in rats and mice as well as humans?	False	False