JÖNKÖPING UNIVERSITY
School of Engineering

LLM-Powered Question Answering on Fault Logs at Scania AB

PAPER WITHIN *Computer Science*
AUTHORS: *Nesly Ibrahim*
TUTOR: *Johannes Oetsch*
JÖNKÖPING *May 2025*

Examiner:	Vladimir Tarasov
Supervisor:	Johannes Oetsch
Scope:	30 credits
Date:	2025-05-11

Abstract

This thesis presents a natural language question-answering system designed for querying structured vehicle fault log data at Scania AB. The system leverages large language models to translate user questions into SQL queries. The core contribution of this work is a pipeline that combines data pre-processing with advanced prompt engineering strategies such as schema grounding, instruction-based prompting, and retrieval-augmented input construction. Experimental evaluation shows that the final implementation, using Anthropic's Claude 3.5 via API, achieves 96% SQL accuracy on a benchmark of domain-specific queries. The study further compares prompt strategies, highlighting that schema+instruction zero-shot prompts and chain-of-thought few-shot prompts perform best. A Streamlit-based chatbot interface with human-in-the-loop validation enables real-time feedback and user trust. Overall, the thesis demonstrates that high-performing, domain-adapted natural language interfaces to structured data can be achieved without fine-tuning by aligning LLMs with clean data and carefully crafted prompts. This approach reduces technical barriers to data access, accelerates decision-making, and provides a scalable foundation for future agentic database systems.

Keywords Natural Language to SQL (NL2SQL), Large Language Models (LLMs), Prompt Engineering, Data Preparation, Isotonic Regression, Schema Linking, In-Context Learning

Sammanfattning

Denna avhandling presenterar ett frågesvarssystem baserat på naturligt språk för att fråga strukturerad fordonsfeldata hos Scania AB. Systemet använder stora språkmodeller (LLM) för att översätta användarfrågor till SQL-frågor. Ett robust systemflöde har utvecklats som kombinerar omfattande datapreparering med avancerade tekniker inom promptdesign, såsom schemagrundning, instruktionsbaserad prompting och retrieval-augmenterad kontext. Den experimentella utvärderingen visar att den slutliga implementationen, med Anthropic Claude 3.5 via API, uppnår 96% noggrannhet i SQL-generering på ett domänspecifikt benchmarktest. Studien jämför även olika promptingstrategier och visar att "schema + instruktion" i zero-shot-läge samt chain-of-thought i few-shot-läge ger bäst resultat. Ett Streamlit-baserat chatbotgränssnitt med människa-i-loopen-validering möjliggör återkoppling i realtid och ökar användarnas förtroende. Sammantaget visar avhandlingen att det är möjligt att uppnå högpresterande, domänanpassade gränssnitt för naturligt språk till strukturerad data utan behov av finjustering, genom att kombinera rena data med noggrant utformade prompts. Detta tillvägagångssätt sänker tröskeln för datatillgång, snabbar upp beslutsfattande och skapar en skalbar grund för framtida agentliknande databassystem.

Nyckelord Naturligt språk till SQL (NL2SQL), Stora språkmodeller (LLM), Promptdesign, Datapreparering, Isoton regression, Schemalänkning, In-kontextinlärning

Acknowledgments

I would like to express my sincere gratitude to everyone who supported me throughout the completion of this thesis.

First, I am deeply grateful to my academic supervisor, Johannes Oetsch, for his continuous support, guidance, and constructive feedback during this project. His expertise has been instrumental in shaping the direction and quality of this work.

My appreciation also goes to Vladimir Tarasov, for his valuable suggestions during thesis lectures that helped in improving the quality of this thesis.

I would like to extend my special thanks to Mohammed Davari, my industrial supervisor at Scania AB, for his invaluable mentorship, technical insights, and encouragement throughout. I am also grateful to Mattias Borg, Manager at Scania R&D, for facilitating this opportunity and supporting the project within the organization.

I would like to thank the entire Vehicle Validation Division for their collaboration and for providing access to the data and expertise necessary for this research.

In addition, I would like to acknowledge the IT support team at Scania, whose timely infrastructure assistance was essential to enable the technical implementation of the project.

Finally, I am truly grateful to my family and friends for their unwavering support, patience, and motivation during this journey.

Contents

Acronyms **VI**

1 Introduction **1**
1.1 Problem Statement . . . 1
1.2 Research Questions . . . 2
1.3 Contributions . . . 2
1.4 Delimitations . . . 3
1.5 Thesis Outline . . . 3

2 Background **4**
2.1 Structured Query Language (SQL) . . . 4
2.1.1 SQL Query . . . 4
2.2 Text-to-SQL Using LLM . . . 5
2.3 Schema Linking . . . 6
2.4 Prompt Engineering . . . 7
2.4.1 Common Prompting Strategies . . . 7
2.5 Retrieval-Augmented Generation (RAG) . . . 9

3 Related Work **10**
3.1 Rule-Based NLIDB Systems . . . 10
3.2 Statistical Approaches . . . 10
3.3 Transformer Models . . . 11
3.4 Pretrained Language Models for Text-to-SQL . . . 12

4 Question Answering Pipeline **14**
4.1 Overview of the QA Pipeline . . . 14
4.2 Data Preparation . . . 15
4.2.1 Data Sources . . . 15
4.2.2 Data Preprocessing . . . 15
4.2.3 Handling Missing Values . . . 16
4.2.4 Unique-Value Extraction . . . 17
4.2.5 Interpolation . . . 17
4.3 Question Answering System Design . . . 19
4.3.1 Initial Experiments with Local LLMs . . . 19
4.3.2 Transition to Claude API . . . 21
4.3.3 Streamlit-Based Chatbot Interface . . . 21
4.3.4 Entity Recognition and Disambiguation . . . 22
4.3.5 Prompt Generation . . . 22
4.3.6 SQL Query Generation . . . 23
4.3.7 Function Calling . . . 23
4.3.8 Query Execution . . . 24
4.3.9 Answer Generation . . . 24

5 Evaluation & Discussion **25**
5.1 Data Quality Impact . . . 25
5.1.1 Evaluation of Interpolation Models . . . 25
5.1.2 Consistency & Reliability . . . 26
5.2 NL2SQL Accuracy and LLM Performance . . . 26
5.2.1 Error Analysis . . . 27
5.2.2 Performance Analysis . . . 27
5.3 Effect of Prompting . . . 27
5.4 Discussion . . . 29

6 Conclusion **31**
6.1 Limitations 31
6.2 Future Work 31

Appendices **38**

Appendix A Prompts **38**
A.1 Basic Zero-shot Prompt 38
A.2 Few-shot Prompt 38
A.3 Schema-Augmented Prompt 38
A.4 Schema + Instruction Prompt (Zero-shot) 39
A.5 CoT + Schema + Instruction Prompt (Few-shot) 39

Appendix B Background on Scania AB **40**

Acronyms

AI Artificial Intelligence.

API Application Programming Interface.

CNN Convolutional Neural Network.

CoT Chain-of-Thought.

CSV Comma-Separated Values.

ECU Electronic Control Unit.

GPR Gaussian Process Regression.

GPT Generative Pre-trained Transformer.

HITL Human-in-the-Loop.

JSON JavaScript Object Notation.

LLM Large Language Model.

MAE Mean Absolute Error.

ML Machine Learning.

NER Named Entity Recognition.

NL2SQL Natural Language to SQL.

NLI Natural Language Interface.

NLIDB Natural Language Interface to Database.

NLP Natural Language Processing.

QA Question Answering.

RAG Retrieval-Augmented Generation.

RAT-SQL Relation-Aware Text-to-SQL.

RDBMS Relational Database Management System.

RMSE Root Mean Squared Error.

RNN Recurrent Neural Network.

Seq2Seq Sequence-to-Sequence.

SQL Structured Query Language.

VIN Vehicle Identification Number.

1 Introduction

Organizations operating in the automotive industry are undergoing a digital transformation, generating huge amount of data. The surge in data availability has opened new opportunities for advanced analytics and intelligent systems to improve operational efficiency and product quality (Sütőová, 2018). Scania, a global leader in the manufacturing of heavy trucks and buses, has increasingly relied on data-driven methodologies to monitor and analyse vehicle performance. New features and systems are continuously tested on pre-production vehicles during validation runs, generating extensive data in the form of vehicle fault logs. Analysis of these logs is critical for diagnosing issues, assessing system reliability for commercial deployment, improving product quality, and guiding future research.

Today, most engineers access databases and analyse data through tools such as Power BI, which require both technical knowledge and domain expertise. While effective, they are time consuming and have large barriers for the wider engineering team, slowing down decision-making and placing additional workload on data experts (Zhao et al., 2024). As a result, even though large volumes of valuable fault data are available, much of its potential remains untapped. An accurate and timely interpretation of the data is essential to ensure vehicle reliability, safety, and performance prior to market release. In light of recent advances in *Natural Language Processing (NLP)* and *Large Language Model (LLM)*, there is a compelling opportunity to develop systems capable of interacting with structured data in natural language.

LLM models, powered by deep learning transformer architectures (Vaswani et al., 2017), exhibit remarkable proficiency in tasks such as text generation, summarization, translation, and knowledge retrieval. Their integration into data analysis workflows unlocks new insights, enables rapid information retrieval and supports deeper analytical exploration. Researchers have explored various *Natural Language to SQL (NL2SQL)* systems that allow users to pose questions in plain language and receive answers directly from databases. Benchmarks such as WikiSQL (Zhong et al., 2017), Spider (Yu et al., 2018), CoSQL (Inui et al., 2019), alongwith models such as Seq2SQL (Zhong et al., 2017) and PICARD (Scholak et al., 2021) have been used to train deep learning models for generating *Structured Query Language (SQL)* queries from text input. However, these models often require large amounts of annotated training data and struggle with domain-specific vocabulary or ambiguous queries.

Due to these limitations, proprietary models developed by OpenAI (Generative Pre-trained Transformer (GPT)) and Anthropic (Claude) have emerged as powerful zero- or few-shot learners, capable of generating SQL queries from natural language without explicit training on the target schema. Techniques like prompt engineering (Sahoo et al., 2024), in-context learning (Rubin et al., 2022) and *Retrieval-Augmented Generation (RAG)* (Lewis et al., 2020) have been found to improve the reliability of LLM outputs. Despite their strengths, these systems still face challenges in understanding schema context and ensuring query correctness. Recent works have begun to incorporate *human-in-the-loop validation, semantic similarity matching* -used to detect paraphrased words- and *schema linking* which maps terms to corresponding elements in the database, to enhance accuracy and reliability in real-world applications (G. Liu et al., 2025). This thesis investigates how LLMs can effectively bridge the gap between natural language queries and structured databases.

1.1 Problem Statement

At Scania, the integration of LLMs with structured vehicle fault data presents several practical challenges. These include translating domain-specific terminology into accurate queries, resolving ambiguous user input, identifying relevant database columns, ensuring explainability, and establishing secure connections to internal data sources. Hence, there is a critical need for a solution that simplifies data access and enables users to gain insights directly from structured vehicle fault data using natural language.

In parallel, the effectiveness of any analytical system also depends on the quality and completeness of the underlying data. One common issue encountered is the presence of missing sensor readings. Among these, odometer values that record vehicle mileage over time are particularly important, as they provide the temporal and usage-based context necessary to interpret fault patterns. Missing odometer data can distort fault timelines and weaken downstream analysis. Addressing this requires robust imputation methods to accurately reconstruct missing values and support reliable time series diagnostics.

1.2 Research Questions

This thesis aims to explore the integration of LLMs for natural language interaction with structured vehicle fault databases in an industrial setting. The following research questions guide the investigation:

RQ1: What preprocessing techniques are necessary to ensure that structured fault log data is clean, consistent, and reliable for natural language querying?

RQ2: How effectively can LLMs generate accurate SQL queries from natural language inputs for domain-specific structured databases?

RQ3: How do prompt engineering strategies affect the accuracy of LLM generated SQL queries in domain-specific applications?

1.3 Contributions

This thesis makes several contributions to the field of *Natural Language Interface (NLI)s* for structured data access, with a focus on fault analysis support within the automotive sector. The work is conducted in collaboration with the Vehicle Validation Division at Scania R&D and addresses practical challenges in deploying LLM-based systems in a real-world industrial environment. Background information on Scania AB and its relevance to this project is provided in Appendix B. The system is built on a static structured SQLite database derived from Scania's historical fault logs, which are filtered and prepared specifically for this study. Naming conventions used in Scania have been implemented to guide schema understanding and improve domain adaptation within prompts and query generation.

1. The study develops a natural language interface that enables non-technical users to interact with structured vehicle fault logs using natural language. It includes a back-end pipeline that performs SQL query generation, query execution, natural language response generation, and a front-end user interface. It supports query explainability and user validation, ensuring transparency and trust in model outputs.
2. The thesis integrates prompt engineering, in-context learning, and semantic linking techniques tailored to domain-specific terminology. In-context learning allows LLMs to adapt to specific tasks using examples provided in the prompt, while semantic linking refers to mapping user queries to relevant database elements. In particular, the system maps ambiguous user inputs to appropriate database fields using a combination of prompt-guided schema descriptions and semantic similarity techniques. These strategies enable accurate and context-aware SQL generation even in complex scenarios.
3. Another contribution of the thesis lies in improving the quality of time-series fault analysis by addressing the problem of missing odometer values. The study applies a *Machine Learning (ML)* component using a regression model to reconstruct the missing values. The model is trained and evaluated on preprocessed datasets using available features such as timestamp and distance metrics.

1.4 Delimitations

The scope of this thesis is limited to a controlled research environment. It does not cover the fine-tuning of language models or large-scale training on domain-specific corpora due to resource and time constraints. Additionally, the system is not designed for real-time production deployment or integration with Scania's live infrastructure; it remains a proof-of-concept built on a snapshot of the test fault log database. The full spectrum of security, data governance and multilingual support are not within the current scope but are acknowledged as future directions.

Furthermore, the ML model used for odometer estimation is limited to interpolation and extrapolation based on available time and distance. The model does not account for external environmental factors, vehicle-specific configurations, or driver behaviour that could affect accuracy of predicted values. The results of the ML component are intended to support analysis and not replace ground-truth measurements in safety-critical systems. The findings can be used for future research in data augmentation for vehicle performance analysis.

1.5 Thesis Outline

This thesis is organized as follows. Chapter 2 provides an overview of the core concepts relevant to this thesis, providing the necessary background to understand the technical foundations upon which the implementation is built. Chapter 3 reviews related work in the field of Natural Language Interface to Database (NLIDB). Chapter 4 outlines the design of the research framework, including data preprocessing, system architecture, and implementation. Chapter 5 presents the evaluation, both quantitative and qualitative assessments of the approach, followed by Chapter 6 which summarizes the contributions of the thesis and outlines directions for further research.

2 Background

This chapter presents an overview of Natural Language Interfaces for querying structured vehicle fault logs at Scania. It begins by introducing SQL as the standard language for querying relational databases and highlights its inherent complexity, which can act as a barrier for domain experts without technical expertise. Building on this foundation, the chapter then introduces the concept of Text-to-SQL, which focuses on translating user questions into executable SQL queries using LLMs. To bridge the gap between user language and database schema, the chapter explores schema linking techniques that map query terms to the correct tables, columns, and values. The role of prompt engineering is then discussed, outlining various strategies that enhance LLM performance in structured query generation. Lastly, the chapter introduces RAG, a framework that dynamically incorporates relevant schema into the LLM's context to improve factual grounding, reduce hallucinations, and support more accurate SQL generation. These components form the theoretical and technical basis for the LLM-powered system developed in this thesis.

2.1 Structured Query Language (SQL)

Automotive systems now incorporate numerous *Electronic Control Unit (ECU)s*, interconnected sensors, and software-driven functionalities, resulting in vast and complex datasets. In industrial contexts, engineers record thousands of fault events during vehicle validation and testing stages. In Scania, these logs are stored in structured relational databases representing valuable assets for quality assurance and debugging processes.

SQL is the foundational language used to manage and query relational databases (Groff & Weinberg, 2002). It allows users to manage, query, and manipulate data stored in tabular formats. SQL supports a variety of operations such as selection (retrieving specific records), projection (selecting certain columns), joining (combining records across tables), filtering (using WHERE conditions), aggregation (e.g., SUM, AVG), grouping (GROUP BY) and sorting (ORDER BY). As a declarative language, SQL enables users to specify what they want from the database, leaving it to the underlying database engine to determine how to execute the request efficiently (Date, 2000).

SQL plays a crucial role in supporting modern data infrastructure, powering back-end services, analytics pipelines, reporting dashboards, and interactive applications. Its syntax and semantics are supported across major Relational Database Management System (RDBMS) such as MySQL, PostgreSQL, SQLite, Oracle, and Microsoft SQL Server (Melton & Simon, 1993). As data-driven decision-making becomes increasingly important, the ability to interact with data through SQL remains a foundational skill for developers, analysts, and researchers alike.

2.1.1 SQL Query

A SQL query is a structured instruction written in SQL to extract specific data from a database. It can be viewed as a formal question posed to the database, specifying what data is needed and the conditions under which it should be retrieved. For example, a basic SQL query might ask, "What are all the fault reports logged in 2024 for the R-series trucks?" which would involve filtering based on the year and vehicle type. This thesis only illustrates SQL through selected examples for clarity; for a comprehensive overview of SQL syntax and functionality, readers are referred to standard database textbooks such as *SQL: The Complete Reference* by Groff and Weinberg, 2002.

A typical SQL SELECT query follows a specific structure:

```
SELECT column1, column2, ...
FROM table_name
```

```
WHERE condition
ORDER BY column ASC|DESC;
```

This query retrieves specific columns (column1, column2, etc.) from a table named “table_name”. The WHERE clause filters the rows based on a given condition, ensuring that only records matching the criteria are selected. Finally, the ORDER BY clause sorts the results according to a specified column, either in ascending (ASC) or descending (DESC) order. The following is a concrete example based on a vehicle fault reporting scenario:

```
SELECT heading, Subject, CreatedDate                                          1
FROM fault_events                                                             2
WHERE Subject = 'R-series' AND strftime('%Y', CreatedDate) = '2024'           3
ORDER BY CreatedDate DESC;                                                    4
```

The above query retrieves specific information from the ‘fault_events’ table about ‘R-series’ trucks in 2024. Line (1) selects the relevant columns - “heading, Subject, and CreatedDate”. Line (2) specifies the table name. Line (3) filters the records where the Subject is ‘R-series’, and the year of the creation is 2024, using *strftime* to extract the year from the timestamp. Finally, line (4) ensures the results are sorted, so that the most recent entries appear first.

More complex queries may involve joins across multiple tables, nested subqueries, conditional logic, and aggregations. Writing such queries requires both domain knowledge of the dataset and technical knowledge of SQL syntax. This complexity becomes a barrier for users unfamiliar with the database structure, which is problematic in industrial contexts where large, interlinked datasets are prevalent.

2.2 Text-to-SQL Using LLM

SQL require in-depth knowledge of the database schema, relationships between tables, and careful construction of filtering conditions. To reduce the complexity of traditional SQL-based querying, researchers have developed NLIDBs that allow users to interact with databases using plain English (or other natural languages). A promising approach within this domain is Text-to-SQL, a subfield of NLP that focuses on translating natural language questions into executable SQL queries (Finegan-Dollak et al., 2018). Text-to-SQL systems aim to make data access more intuitive for users without technical background. Given a relational database D and a natural language question q_{nl} to D, translating natural language to SQL is to find a SQL query q_{sql} that correctly answers q_{nl}. In recent years, NL2SQL has emerged as an active area of research in both the database and NLP communities (Kim et al., 2020).

Text-to-SQL requires understanding both the semantics of the user’s query and the structure of the underlying database. This involves identifying relevant tables and fields, inferring the appropriate query logic (e.g. joins, filters, aggregations) and constructing a valid SQL syntax that answers the original question. To benchmark progress in this area, Finegan-Dollak et al., 2018 introduced datasets that evaluate model performance across diverse database schemas, highlighting the difficulty of generalizing across different data structures. Building on this, a more recent work by Tan et al., 2024 emphasized the importance of schema linking and the use of pre-trained embeddings to enhance the model’s semantic understanding and ability to map query terms to the correct database elements. Schema linking techniques used in this thesis are discussed in detail in Section 2.3. Effective schema linking improves the model’s ability to ground user queries in the database structure, especially when users refer to database fields using synonyms or domain-specific language. It is supported by pre-trained language models, such as BERT (Devlin et al., 2019) or T5 (Raffel et al., 2020), which are trained on large corpora and then fine-tuned for *Question Answering (QA)*, allowing them to capture nuanced meanings and relationships. In Text-to-SQL tasks, such models help encode both the natural language input and the database schema in a shared representation space, facilitating more accurate query generation.

A new direction in Text-to-SQL tasks involves the use of open-source LLMs such as Mistral (Jiang et al., 2023) , LLaMA(Touvron et al., 2023) and proprietary models like GPT (Radford et al., 2018) and Claude (Bai et al., 2022). Models like CodeLLaMA (Rozière et al., 2023) have shown strong performance on SQL generation tasks due to their specialized training on structured data, programming languages, and task-oriented dialogues. They exhibit better syntactic accuracy and alignment with schema-based contexts. The effectiveness of these models is further enhanced through prompt engineering, a technique used to guide the model's output by carefully designing its input instructions (discussed in Section 2.4). However, initial experiments with CodeLLaMA revealed that, without domain-specific fine-tuning, the model produced hallucinated table names and logic errors. The need to prepare a large training dataset for fine-tuning exceeded the scope and timeline of this thesis.

2.3 Schema Linking

Schema Linking in Text-to-SQL systems connects words or phrases in a natural language question to elements in the database schema such as table names, column names, etc. When users ask a question in plain language, they often do not refer to the database elements exactly as they are named in the schema. This makes it difficult to correctly identify the relevant columns, especially when synonyms, abbreviations, or domain-specific terminology are used. Inaccurate mapping can result in SQL queries that refer to incorrect or non-existent schema elements leading to wrong answers. Schema linking helps the system map user terminology to the correct database fields, making SQL query generation more accurate.

Example. Given the question: *"List all faults from R-series vehicles reported in 2024"* and a database schema with a table `fault_events` and columns like `Subject`, `CreatedDate`, and `heading`, the system must determine that:

- "faults" maps to `heading` or a fault-related column,
- "R-series" is a value in the `Subject` column,
- "reported in 2024" refers to filtering on `CreatedDate`.

Types of Linking: In NL2SQL systems, different types of linking play a crucial role in bridging the gap between user queries and database structures. Although this thesis uses an LLM (Claude) to directly translate natural language queries into SQL, the process implicitly involves several types of schema linking:

- *Schema Element Linking:* Schema element linking involves identifying which parts of the user question refer to database tables or column names. For instance, a term like "vehicle" may need to be linked to a column named Subject.
- *Value Linking:* Value linking refers to recognizing when a term in the query is a literal value from the database, such as mapping "R-series" to entries under the Subject column.
- *Synonym Linking:* Synonym Linking deals with understanding that different words with similar meanings may refer to the same database field. Terms like "logged", "reported", and "recorded" can all relate to a date column like `CreatedDate`.

Techniques Used: In this thesis, several techniques were employed to link natural language tokens with database schema elements and values. These methods help to accurately interpret the user intent and map it to the appropriate SQL structure.

- *String Matching:* To handle minor misspellings or inconsistent terminology, fuzzy matching (e.g., Levenshtein distance) is used to compare user input with known column names and values. This helps resolve inputs like “Jacb” to “Jacob”. In this thesis, fuzzy matching is implemented using the `RapidFuzz` Python library[1], which provides fast and efficient similarity scoring.
- *Entity Disambiguation:* If a keyword matches multiple database fields (e.g., a name appearing in both “Reporter” and “Responsible” columns), it prompts the user for clarification. Contextual knowledge is used to detect and disambiguate such conflicts.

These combined techniques reduce the need for users to reference exact database terminology, thereby enhancing overall robustness and usability. Schema linking is crucial for ensuring semantic fidelity between a user’s intent and the resulting SQL query. Without accurate schema linking, even powerful language models may generate syntactically correct but semantically incorrect queries.

2.4 Prompt Engineering

Prompt engineering refers to the design of input prompts to guide LLMs toward generating desired outputs. Since LLMs like GPT and Claude are sensitive to the structure and content of the input, the formulation of the prompt can significantly influence the model’s response quality (Zhang et al., 2024). Effective prompt engineering guide the model with tailored inputs to achieve the desired behavior without modifying the model weights or relying on large-scale fine-tuning.

Prompting has gained prominence in Text-to-SQL tasks, where the aim is to generate syntactically and semantically correct SQL queries based on a user’s natural language questions. Rather than hardcoding logic or retraining models, prompt engineering enables zero-shot or few-shot learning, making it possible for general-purpose LLMs to work with new domains and database schemas.

A core mechanism enabling prompt engineering is *in-context learning* (Brown et al., 2020). In this paradigm, the model learns the task by conditioning on examples and instructions provided entirely within the prompt. The LLM takes advantage of these demonstrations and patterns as part of its contextual input and generates responses accordingly. This ability allows flexible adaptation for Text-to-SQL generation by simply modifying the input prompt, thereby significantly lowering the barrier for deploying LLMs in domain-specific applications.

2.4.1 Common Prompting Strategies

Several prompting strategies have been developed to enhance the performance of LLMs in downstream tasks by shaping the input context in ways that align the model’s prior knowledge with task-specific constraints (Zhang et al., 2024). These strategies exploit the model’s learning capabilities and enable zero-shot or few-shot generalization without gradient updates. Each strategy varies in how much task-specific supervision it embeds into the prompt:

- *Zero-shot Prompting:* It operates under the assumption that pretrained LLMs have already learned enough generalizable knowledge to perform a task with only minimal task instruction. The prompt includes only the input question and an instruction (e.g., “Convert this question to SQL”), and the model must infer the solution without any examples. This approach is suitable when labeled examples are unavailable or minimal latency is required.
- *Few-shot Prompting:* Few-shot prompting builds on the model’s ability to generalize from a small number of in-context demonstrations. The prompt includes multiple annotated input-output pairs

[1]https://maxbachmann.github.io/RapidFuzz/

(e.g., natural language questions paired with SQL queries), followed by a new test input. The model learns the mapping pattern and generate an analogous output for the new input. This method reduces the reliance on fine-tuning and is particularly effective for pattern-based tasks like Text-to-SQL (Brown et al., 2020).

- *Schema-Augmented Prompting:* Schema-augmented prompting addresses the domain adaptation challenge in structured data tasks. It augments the prompt with explicit metadata about the database schema, such as table names and column descriptions. By exposing this structural information, the model can better disambiguate terms in the user query and map them to corresponding database fields, improving semantic alignment (Rajkumar et al., 2022).

- *Instruction-based Prompting:* Instructional prompting involves providing the model with clear instructions. It encodes the task objective and behavioural expectations directly into the prompt. Rather than relying solely on examples, it positions the LLM as an agent with a defined role (e.g.,"You are an expert SQL assistant") and guides its behaviour through declarative or imperative instructions. This narrows the scope of model behaviour and ensure it understands the task objective. It improves controllability and reduce hallucination, particularly in multi-step or constrained generation tasks.

- Chain-of-Thought *(CoT) prompting:* Also known as *Multistep prompting*, this is a prompt engineering strategy in which the model is guided to reason through intermediate steps before producing the final answer. Instead of asking an LLM to perform an end-to-end task in a single pass, the prompt encourages the model to first decompose the task into sub-problems or generate reasoning traces that lead to the correct solution (Wei et al., 2022). The output of each step can feed into the next, enabling greater control, interpretability and reliability. This is useful in complex tasks that require logical inference, multi-hop reasoning or procedural understanding like generating SQL queries from vague or compound questions. LLMs often struggle with compositional queries or tasks that require reasoning across multiple dimensions.

  For example, consider the user query: *"List all faults reported by Jacob in 2024 for the R-series."* Instead of prompting the model directly to generate SQL, the following method is used:

  1. *Identify schema elements:* "Which columns in the table 'fault_events' are relevant for answering the question?"
     Expected Output: Subject, Reporter, CreatedDate
  2. *Extract conditions:* "What are the filter conditions implied by the query?"
     Expected Output: Subject = 'R-series', Reporter = 'Jacob', Year(CreatedDate) = 2024
  3. *Generate SQL:* "Write a SQL query over the fault_events table using the extracted filters."
     Expected Output:

     ```
     SELECT heading, Subject, CreatedDate
     FROM fault_events
     WHERE Subject = 'R-series' AND Reporter = 'Jacob'
     AND strftime('%Y', CreatedDate) = '2024';
     ```

These prompting strategies serve as modular techniques for controlling LLM behaviour in various deployment settings. Their effectiveness often depends on the complexity of the task, the availability of examples and the specificity of the domain information. A combination of schema and instructions allows the model to generate valid SQL queries without retraining on the specific schema. Prompt engineering is thus a lightweight and effective alternative to fine-tuning in the context of domain-specific applications.

2.5 Retrieval-Augmented Generation (RAG)

The capability of LLMs to generate contextually rich responses has enabled numerous breakthroughs in NLP. However, a limitation is their inability to access external, real-time, or domain-specific knowledge unless explicitly encoded during training. This becomes problematic in specialized applications like Text-to-SQL over domain-specific datasets where model outputs must be tightly grounded in external facts, structured data, or changing schema. To address this, retrieval-augmented approaches have been developed, combining parametric language models with non-parametric memory components such as document indices or knowledge bases. One influential instantiation of this idea is the RAG framework proposed by Lewis et al., 2020, which integrates a retrieval step directly into the generation process to condition the model on relevant external information. This approach has shown to improve factual accuracy and specificity of generated text while reducing open-domain hallucinations (Lewis et al., 2020). By grounding responses in retrieved evidence, RAG provides a dynamic knowledge source that can be updated or expanded without retraining the underlying model.

At its core, RAG is a two-stage framework:

1. *Retrieval*: Given a user query, a retriever module fetches relevant documents, database schema or other context from a knowledge base.
2. *Generation*: The retrieved information is passed as part of the prompt to the LLM, which conditions its response on both the original query and the retrieved content.

This enables the model to generate answers that are both fluent and factually grounded, even when the answer is not part of its pre-trained parameters. In the context of Text-to-SQL, RAG is valuable for several reasons. The automotive fault logs used in this thesis often contain specific terminology and fault classification systems. Schema grounding can be difficult for LLMs in unfamiliar domains or over large complex schemas. By retrieving schema descriptions and glossary terms, RAG helps the model to understand and align with these terms. By explicitly supplying the LLM with relevant table names, column values, etc., it reduces the chances of hallucinating non-existent columns or misinterpreting user intent (Ziletti & D'Ambrosi, 2024).

RAG supports better query disambiguation, improved schema grounding, and more accurate SQL generation across diverse queries. For example, when a user asks, "What faults were reported by Jacob?", the retriever checks multiple columns (e.g., Reporter, Responsible) to determine all potential matches and then passes the context to the LLM to generate a clarifying prompt or resolve ambiguity based on retrieved metadata.

3 Related Work

This section reviews the evolution of Natural Language Interface to databases, particularly in the context of querying structured data. The discussion traces the development of techniques from early rule-based systems through statistical and neural models and into modern methods utilizing transformer-based language models, prompt engineering, and RAG. Each subsection explains the respective approach, its achievements, limitations in the context of automotive fault diagnosis, and how this thesis builds upon or addresses those shortcomings.

3.1 Rule-Based NLIDB Systems

NLIDBs have long been a research goal in *Artificial Intelligence (AI)*. Early NLIDBs relied on manually crafted rules, grammars and pattern matching to map questions to database queries. These were simple and often effective on their narrow domain, using hard-coded phrase patterns to trigger certain SQL. The advantage of this approach was that it bypassed complex parsing. As long as the user's question fits a known pattern, the system could produce a correct SQL statement (Androutsopoulos et al., 1995). However, these systems had severe limitations. For instance, a slight rewording or an unforeseen combination of conditions would confuse the pattern matcher, yielding incorrect or no results. In the domain of vehicle fault data, a purely template-based system would require anticipating every way a user might phrase a query, an impractical burden.

Despite their brittleness, these early systems established important paradigms. Interactive disambiguation was explored as a remedy for ambiguity; for example, *Natural Language Interface for Relational Databases* (NaLIR) developed by Li and Jagadish, 2014, introduced a pipeline that parses a user's question into a formal intermediate query tree, then asks the user to verify or refine this interpretation before executing it. The authors of NaLIR demonstrated that by involving users in a limited dialog, the system could correctly translate complex questions involving aggregation, joins, and nested subqueries into SQL across different domains. However, NaLIR and similar systems still relied on a hand-crafted parsing grammar and domain lexicons. In highly technical domains like vehicle diagnostics, where terminology varies, maintaining such rules becomes prohibitively expensive. Moreover, constant back-and-forth with the user may be cumbersome in practice, as the user might find it disruptive to answer clarification questions for each query. This thesis builds on the idea of user validation from NaLIR but applies it to minimize user effort by utilizing the strong language understanding of large pretrained models to interpret queries in one shot. Rather than involving the user during query parsing, the system first generates SQL and then allows the user to validate the query's output in limited guided ways.

3.2 Statistical Approaches

To overcome the rigidity of rule-based methods, researchers began exploring data-driven approaches. One of the early methods was the use of *Inductive Logic Programming* (ILP), to learn semantic parsers for database queries directly from paired examples (Zelle & Mooney, 1996). They employed ILP techniques to induce logic-based rules, mapping natural language questions to database queries. This methodology reduced reliance on manually crafted grammars by enabling the parser to automatically generalize linguistic patterns from training data. When evaluated in U.S geography domain, this data-driven ILP-based approach demonstrated greater flexibility and robustness in handling diverse and complex query variations compared to traditional grammar-based methods. However, applying such parser effectively to structured vehicle fault data would require a sizable annotated corpus of natural language questions paired with accurate SQL queries, a resource that is often unavailable in practice.

Research in semantic parsing saw a resurgence with the advent of neural networks. These models used encoder-decoder architectures to learn direct mappings from natural language to SQL. The introduction of *Sequence-to-Sequence (Seq2Seq)* learning and attention mechanisms enabled the decoder to focus on relevant input tokens. A Recurrent Neural Network (RNN)-based encoder-decoder could learn to generate logical forms, when aided by attention to focus on relevant parts of the question (Dong & Lapata, 2018). Dong and Lapata, 2018 introduced a coarse-to-fine decoding strategy for semantic parsing, where the model first predicts a high-level sketch of the query and then fill in details. This reduced the decoder's search space and improved accuracy on standard benchmarks like WikiSQL. The limitation of these neural models is, if trained naively, can make mistakes such as producing syntactically invalid SQL or misaligning a condition value to the wrong column.

A significant milestone in the development of Text-to-SQL systems was the introduction of neural network models and large-scale benchmark datasets that enabled training and rigorous evaluation. WikiSQL (Zhong et al., 2017) was introduced in 2017 as one of the first massively annotated NL-SQL datasets, with 80,654 training pairs. The queries are relatively simple, but it allowed the data hungry neural models to be trained for the first time. Despite its size, WikiSQL was limited by its synthetic construction; natural language questions were paraphrased from SQL templates, reducing linguistic diversity and limiting real-world generalization (Yavuz et al., 2018). Zhong et al., 2017 also proposed the Seq2SQL model with reinforcement learning achieving good accuracy by using policy gradient to refine the decoder's SQL predictions.

The release of the Spider dataset (Yu et al., 2018) in 2018 marked a shift toward more realistic and challenging Text-to-SQL evaluation. Spider contains 10,181 questions over 200 different databases covering diverse domains, with SQL queries that often involve multiple tables, nested subqueries, and other complex SQL phenomena. The dataset enforces schema generalization by ensuring that test databases are unseen during training. This design addressed prior concerns about overfitting to query templates, as raised by Finegan-Dollak et al., 2018. Its impact is also reflected in the rise of models that explicitly model schema structure, such as Relation-Aware Text-to-SQL (RAT-SQL) and IRNet, which were developed in response to the challenges posed by Spider.

3.3 Transformer Models

The introduction of the Transformer architecture changed the landscape of NLP, including Text-to-SQL systems. Introduced by Vaswani et al., 2017, transformers use self-attention to capture long-range dependencies in sequences, enabling powerful modeling of natural language compared to earlier RNN or Convolutional Neural Network (CNN)-based models. This breakthrough led to the development of large-scale pretrained language models like BERT (Devlin et al., 2019), RoBERTa (Y. Liu et al., 2019), T5 (Raffel et al., 2020), and GPT (Radford et al., 2018), all of which were rapidly adopted for semantic parsing tasks.

Fine-tuning these models for Text-to-SQL tasks led to significant improvements in accuracy and generalization. However, a challenge in Text-to-SQL is effective schema linking—i.e., grounding words in the natural language query to specific columns, tables, or values in the database. IRNet (Guo et al., 2019) addressed this by introducing an intermediate representation to decouple the NL-to-SQL mapping into two steps: first map the NL question to a tree of logical operators, and then deterministically convert that tree to SQL. This intermediate tree used nodes for representing operations like JOIN or FILTER, which made it easier to ensure all necessary joins are present and to systematically expand nested queries. While this improved structural correctness and compositional generalization, IRNet still relied heavily on exact string matching for schema linking to link question words to schema elements. For domains where terminology might not exactly match column names, pure string matching is insufficient. To address this, Ghosal et al., 2019 proposed encoding the database schema as a graph and using *Graph Neural Networks* (GNNs) to jointly model the schema and the query. This allowed the model to learn not only the names of the tables and columns but also their relationships, improving join selection and column resolution. *Relation-Aware Text-to-SQL* (RAT-SQL) by Wang et al., 2020 further advanced this approach by introducing relation-aware self-attention, indicating whether two tokens from question or schema are in certain relations. It could effectively perform schema linking and focus attention on the relevant parts of the schema for a given question and achieved strong generalization. These methods significantly improved performance on Spider, a benchmark to test generalization to unseen schemas.

However, these systems have practical limitations in industrial settings. RAT-SQL, while highly accurate, is a complex neural architecture requiring substantial training data to tune all the attention parameters. Another prominent approach is BRIDGE, developed by Lin et al., 2020, which leverages pre-trained language models to bridge the semantic gap between text and schema. BRIDGE employs intermediate representations and contextualized embeddings to link natural language to schema elements, outperforming earlier rule-based approaches. Schema linking was further boosted by works like TaBERT (Yin et al., 2020) which pre-trains models jointly over textual data and tabular schema. TaBERT's contextualized table representations have proven effective in improving schema linking and value linking where the model must relate question words to actual data content. Rather than developing a new pretraining from scratch, this thesis supplement the question with retrieval augmentation. While TaBERT bake in certain knowledge during pretraining, this method can pull in fresh, specific information at query time.

3.4 Pretrained Language Models for Text-to-SQL

The emergence of LLMs such as GPT, Claude and Codex marked a significant shift in Text-to-SQL research. These models have demonstrated remarkable capabilities in generating valid SQL queries even in zero-shot or few-shot settings without requiring task-specific fine-tuning. This has enabled the development of flexible and high-performing systems for NLIs to structured data, where retraining on every new schema is impractical. Recent studies, such as SolidSQL (G. Liu et al., 2025), have shown that schema-aware prompting can significantly improve the accuracy of SQL generation. This is achieved by serializing the schema by representing table and column names in a structured format and appending it to the input prompt. Models like Codex showed strong Text-to-SQL ability when simply provided the schema and question (Rajkumar et al., 2022).

Another effective technique is to provide few-shot examples of natural language questions and their corresponding SQL queries within the prompt. These in-context examples demonstrate the task format and logic to the model. Studies have shown that even a handful of examples can boost accuracy significantly, often matching or surpassing some fine-tuned models (Rajkumar et al., 2022). Using examples with a similar schema or SQL structure to the target question, yields better performance (Nan et al., 2023). These examples may be statically embedded or dynamically retrieved from a library of past queries at runtime.

Instructional prompting, which explicitly frames the task (e.g., "Translate the question below into an SQL query using the schema provided"), has also proven effective, especially for dialogue-oriented models like Claude. Without such guidance, LLMs may generate verbose explanations or incorrectly formatted output. The SQL-PaLM (Pathways Language Model) framework presents a comprehensive approach to advance Text-to-SQL using LLMs, combining few-shot prompting and instruction fine-tuning (Sun et al., 2023). It explores execution-guided consistency decoding, synthetic data augmentation and schema selection techniques to enhance performance. Beyond structural awareness and exemplar-based learning, recent advances have explored how to improve LLM reasoning for more complex queries. Tan et al., 2024 emphasize the importance of carefully structured prompts, demonstrating that small variations in format or phrasing can lead to significant differences in output quality.

The rapid development of prompting methods has led to the development of numerous effective prompting strategies. One influential approach is Chain-of-Thought (CoT) prompting (Kojima et al., 2022) which improve LLMs' reasoning ability by producing intermediate steps before predicting a final answer. It encourages step-by-step reasoning, such as thinking aloud about which tables and joins are needed before formulating SQL query. CoT prompting has been shown to significantly improve LLMs' ability to handle multi-hop database questions (Tai et al., 2023). Another related line of work involves decomposing a complex question into simpler sub-questions. Each is translated into partial SQL logic and then composed into a full query. Eyal et al., 2023 demonstrate that such decomposition can be handled entirely within the prompt. *Dynamic Interaction Network for Text-to-SQL* (DIN-SQL) extends this further by incorporating execution-based self-correction, which detects and repairs incorrect SQL by analyzing query outcomes. These reasoning-based methods improves robustness in zero-shot scenarios with complex schema interactions (Pourreza & Rafiei, 2023).

Empirical studies on challenging benchmarks (e.g., Spider) have validated the effectiveness of these prompting strategies. Carefully engineered prompts enable large LLMs to attain accuracy close to fully trained Text-to-SQL models. Nan et al., 2023 report that careful prompt design alone can surpass fine-tuned models in execution accuracy. Similarly, Gao et al., 2024 achieved 86% accuracy on Spider using prompting alone, a performance level comparable to that of the best supervised models. These results demonstrate that prompting can unlock strong SQL generation capabilities from proprietary models like Claude without any training.

In industry contexts, where database schemas change frequently, prompt-based methods offer substantial advantages. The LLM serves as a SQL generation engine that is fed with the current schema and context each time. This prompt-based method avoids the costly retraining when the schema changes. However, as prompts become longer and more complex, inference latency and token limits become bottlenecks. To mitigate this, many systems incorporate lightweight retrieval modules to dynamically select only the most relevant schema elements. With LLMs accepting ever longer inputs and improved prompt techniques, this approach is becoming a viable solution for NLIDBs.

In this thesis, these strategies are integrated into a unified Text-to-SQL framework tailored for querying Scania's vehicle fault database. The approach combines prompt engineering with a retrieval mechanism that injects database-specific schema information like column names, table structure, and known value sets into the prompt dynamically. This balances the power of general-purpose LLMs with the specificity needed for domain-grounded tasks. The model is not trained on Scania's vehicle fault schema but is able to understand and query it effectively due to contextual prompt design.

4 Question Answering Pipeline

This section outlines the methodology adopted to design and implement a robust QA pipeline for the analysis of vehicle fault data. This methodology is grounded in the principles of information retrieval, NLP, and structured data integration.

4.1 Overview of the QA Pipeline

The overall system is designed to enable natural language querying of a vehicle fault database by translating user questions into accurate SQL queries using an LLM. The architecture consists of two broad components: a data preparation pipeline and a QA pipeline for NL2SQL conversion. In the data preparation stage, raw data extracted from the database is preprocessed to remove errors and inconsistencies, missing values are interpolated, and key information is extracted into a structured format. The output of this stage is a refined fault database and a JavaScript Object Notation (JSON) file capturing unique values of important fields.

In the QA stage, the user's query is processed through a pipeline of prompt construction, LLM inference, and result verification. The system first retrieves relevant schema information or values from the prepared data, which is then combined with a carefully engineered prompt template. The prompt is dynamically constructed to include the user's question and the auxiliary context so that the LLM has the information needed to generate a correct SQL query. The LLM which is a cloud-based model, Anthropic Claude, processes this prompt and produces a SQL query, intended to answer the user's question. The generated SQL query is executed on the cleaned database. The resulting response is then returned to the LLM to generate a natural language response for the user. The user interacts with the QA system through a Streamlit web application that mimics a chatbot interface.

Figure 1 illustrates the system architecture, where raw data is preprocessed into a cleaned database. User queries are processed through the Question Answering pipeline, and the results are returned as natural language answers.

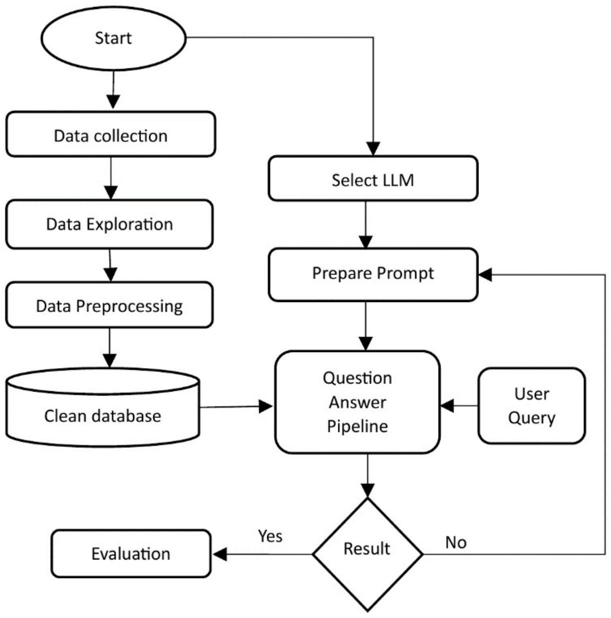


Figure 1. Flowchart showing workflow.

4.2 Data Preparation

The first phase of the project focused on preparing the vehicle fault data for effective use in the querying system. Raw data often contains various quality issues, e.g., missing entries, inconsistent formatting, and irrelevant information that can impair analysis. Therefore, extensive data preparation was performed to ensure a high-quality database. These steps conducted manually in this feasibility study could be automated in a future production pipeline. This section details the data cleaning steps, the extraction of unique values into a JSON file, and the handling of missing data through interpolation. This thesis also compares the four interpolation techniques examined: *linear regression (linear interpolation), spline interpolation, Gaussian Process Regression (GPR), isotonic regression*, and explains why isotonic regression was ultimately selected for this task.

4.2.1 Data Sources

The data used in this thesis originate from fault event logs collected from Scania test trucks. These logs are stored and managed in a cloud-based enterprise data warehouse, Snowflake, which serves as the central repository for high-volume vehicle diagnostics and performance data. The analysis relies on two primary data sources extracted from the database: (1) vehicle fault event data and (2) vehicle odometer readings. Each fault event includes a rich set of 28 metadata fields such as event id, fault title (heading), vehicle identifier (subject), event type, project name, timestamps (create date, failure date, solved date, closed date), personnel-related fields (reported by, responsible), organizational groupings (reported group, responsible group), hierarchical fault classification (main group, sub group description), status, etc. Another extraction was used to bridge different vehicle identification systems, listing vehicles by both their Vehicle Identification Number (VIN) and an External Equipment Reference number with their odometer readings over time.

4.2.2 Data Preprocessing

For this project, historical data from heterogeneous sources were exported to Excel spreadsheets and Comma-Separated Values (CSV) files. All files are then loaded into the analysis environment using Pandas, taking care to preserve data formats. Data cleaning involved multiple passes and techniques to ensure that it is accurate and consistent. The CSV files with odometer data were first pre-processed and merged into a single consolidated dataset. Each file contained time-stamped odometer readings for a set of vehicles, but with slightly different schemas. To harmonize these, all non-essential columns are removed, retaining only the core fields needed for analysis: the vehicle's reference number, the time-stamp and the odometer value. Next, the data types are standardized: the time-stamp field in a date-time compatible string format, and the odometer value field is ensured to be numeric, converting the comma separators to dot and rounding to a consistent number of decimal places.

The combined odometer dataset initially contained 16,260 records. After applying a deduplication step to eliminate exact duplicates, the dataset is reduced to 15,732 unique entries. Since event data and odometer data used different identification keys for vehicles, it is crucial to reconcile the naming conventions before merging. In event records, vehicles are identified by a short code name, while odometer readings are tagged by External Equipment Reference number. A separate identifier, the vehicle chassis number, which is a unique substring of the VIN, was available as a common link between the datasets. In practice, the chassis number for each vehicle is standardized to a 7-character code corresponding to the last 7 characters of the VIN. To avoid inconsistencies, a series of string cleaning and normalization steps are performed on vehicle names and VINs. All vehicle names are treated in a uniform case and VIN numbers are handled as strings to preserve any leading zeros and to maintain consistent length.

With clean and consistent identifiers, the next step is to integrate the event data with the odometer readings. This is done by constructing a mapping table that links the event's vehicle name to the corresponding odometer reference. The chassis number, an intermediate key is used for this alignment. A multi-step data integration process involving three separate datasets, as shown below, is performed to map the vehicle's fault data with mileage.

1. *Odometer Data:* Contains External Equipment Reference, VIN and corresponding distance readings.
2. *Chassis-Vehicle Mapping:* Maps Chassis Numbers to Vehicle Names.
3. *Fault Events Data:* Contains Vehicle Names and detailed fault records.

The vehicle names are then appended to the odometer dataset using the chassis number. This aligned each odometer reading with the vehicle name used in the event logs. A left join on the odometer data is done so that odometer entries for vehicles that never had any reported events would remain in the data with a null vehicle name, whereas all odometer entries for vehicles present in the chassis vehicle mapping dataset would now carry the correct vehicle name. After a minor cleanup of redundant columns, the dataset now has three columns namely vehicle names, time-stamp and odometer value. But it had many entries per day at different times for every vehicle. To facilitate merging by date, the latest odometer value for each vehicle on each calendar date is selected (since the readings are cumulative and non-decreasing).

From the fault event dataset, the "Subject" column (contain vehicle name) and "Date of failure" column is taken as the key temporal marker for when the failure occurred. The date fields in this dataset are also converted to proper date-time objects and normalized to ensure that the merges are accurate. The merging of the event data with this daily odometer data is also done as a left join, ensuring that every failure event remained in the result, with an odometer value attached where available. The new column is named as "Actual odometer" to clearly indicate that it represents the vehicle's real odometer reading at the time of failure.

4.2.3 Handling Missing Values

Missing data is another significant issue that affects data quality. Many records had missing values in one or more fields. Rather than simply discarding all records with missing values, which can introduce bias or significantly reduce dataset size, they are imputed where possible. During the mapping process, any rows with missing critical identifiers such as the VIN or the equipment reference are dropped because they could not contribute to a reliable vehicle mapping. Likewise, if the odometer data is missing any vehicle names which are present in the fault event data, it would not be mapable to odometer data. Such cases are kept in the event dataset for completeness, but would result in no odometer match in the combined data.

After merging the events with the odometer readings by date, the extent of missing odometer values for the events is evaluated. It is found that a substantial number of events did not have an exact odometer reading on the failure date. Some vehicles did not report a reading when the fault occurred, or some vehicles did not have telemetry data in the odometer dataset. In fact, more than half of the failure events (approximately 1,918 out of 3,574 events) had a missing odometer value after the merge. To address this, a strategy was implemented to fill in these missing readings in a reasonable way. For events associated with vehicles entirely absent from the odometer data (meaning the vehicle was never recorded in the telemetry files), the odometer value at failure is treated as 'not available' and set to 0. For events where the vehicle is tracked on the odometer data but do not have a reading on the exact failure date, the odometer value is estimated by interpolation.

Since odometer readings are cumulative and non-decreasing over time, each vehicle's daily odometer time series is interpolated using a monotonic method. An isotonic regression model is fitted per vehicle to its daily odometer points, providing a smooth non-decreasing function of mileage over time. This model is then used to predict the odometer reading on the exact date of failure if it lays between known readings or extrapolate slightly if the failure occurred just before the first recorded reading or just after the last, assuming a steady rate of mileage accumulation. The interpolated values are inserted for the missing cases, ensuring that every event in the final dataset has an associated odometer reading; either the actual recorded value or the estimated value. All such inferred values are flagged as such to be transparent in the analysis. Beyond these, other types of missing values in the event data like in dates or personnel-related fields were filled with 'unknown' to maintain dataset completeness without impacting downstream processing. A detailed description of the interpolation procedure is provided in Section 4.2.5.

4.2.4 Unique-Value Extraction

The unique values of key fields from event data are extracted and stored in a JSON file. This JSON file serves as a lightweight knowledge base, which is later used by the QA system for prompt augmentation. Lists of distinct entries for fields that users are likely to refer to in queries or that are helpful for the LLM to know, are compiled. Examples include a list of all fault types, vehicle names, and other personnel-related fields. By storing these in a structured JSON, it is easier to retrieve and inject this information into prompts to validate user inputs.

The rationale for creating this file is two-fold. Firstly, it allows mapping the user terminology to the actual database terms. If a user asks a question using a specific term (e.g., a fault name like "engine overheating"), the system can consult the JSON file to see if"engine overheating" matches a known fault description in the data. This helps LLM ground its results to actual data and addresses a common limitation: without grounding, they might hallucinate or guess schema labels. By giving the model a list of actual values to choose from, the chances of producing a fault name that does not exist in the database are reduced. In essence, the JSON file acts as a simple lookup-based retrieval mechanism, ensuring that the LLM's output remains consistent with the real data content.

Secondly, JSON provides a form of schema documentation for prompt construction. Rather than including the entire database or table in the prompt, only the relevant slice is included. For example, if the user query mentions "SONNY", the LLM can easily identify if the user is referring to any person or vehicle name. This JSON file is generated programmatically by querying the distinct values of each relevant column in the cleaned dataset. It not only aids the LLM, but also provides transparency to users about what values exist in the data. For instance, if a query contains a vehicle name that does not exist in the dataset, the interface can detect the mismatch, alert the user to a potential typo, and prevent the generation of an invalid query. This design choice proved useful in maintaining alignment between user queries and actual database content.

4.2.5 Interpolation

A significant challenge in the preparation of the database was handling missing odometer readings. Instead of dropping those records, the missing values are estimated using the available data. Four different regression techniques are explored for interpolation:

- *Linear Regression:* filling missing values by assuming a linear trend between known data points.
- *Spline Interpolation:* using piecewise polynomial (typically cubic) splines to interpolate a smooth curve through the known data points.
- *GPR:* modelling the data with a gaussian process to predict missing values, providing a probabilistic interpolation with uncertainty estimates.

- *Isotonic Regression:* fitting a monotonically increasing (or decreasing) function through the data, used if an underlying monotonic trend is expected, and then interpolating missing points along that fitted function.

Each of these methods has different assumptions and characteristics. They are qualitatively compared on the data to decide which is the most suitable before integrating the chosen method into the final pipeline.

Linear interpolation is used as a baseline due to its simplicity. It takes two known data points and fills in the missing value assuming a straight line between them. This method is easy to understand and often effective for small gaps in data or approximately linear processes. However, it can be inaccurate if the true underlying relationship is non-linear or if the gap is large, as it cannot capture any curvature or dynamics in the data beyond a constant rate of change. If on April 1 the odometer recorded 50,000 km, and 50,100 km on April 3, linear interpolation estimates the April 2 value as 50,050 km. This estimate will be reasonable if the vehicle is driven consistently each day. However, if there are sudden variations in vehicle usage (e.g., no driving or a long trip on a specific day), linear interpolation might not accurately reflect the true mileage progression.

Spline interpolation provides a more flexible approach. A spline is a piecewise polynomial function that passes through all the known data points and is typically constrained to ensure smoothness at the joins. Most commonly, cubic splines are used, fitting third-degree polynomials between points with smooth transitions ensured by matching first and second derivatives. The benefit is that it can model curves more accurately by avoiding the extreme oscillations of high-degree polynomial fits using low-degree polynomials in a piecewise manner. But a limitation is that cubic splines can oscillate in regions with sparse data. For example, if the odometer readings of a vehicle on April 1 is 50,000 km, 50,300 km on April 4, and 50,600 km on April 7, but no readings are available for days in between, a spline interpolation will create a smooth curve, estimating values for the missing dates.

GPR is a non-parametric, probabilistic model which assumes that the data is drawn from a multivariate Gaussian distribution. A Gaussian process is a distribution over functions, defined by a mean function and a covariance (kernel) function to handle non-linear trends, provide smooth estimates, and quantify the uncertainty of its predictions (Rasmussen & Williams, 2006). GPR has an interpolating property as well as a probabilistic nature, which means that it not only provides an estimate but also a confidence range. GPR can capture complex, non-linear trends in the odometer trajectory without specifying a fixed functional form. For example, if the vehicle's odometer readings on April 1 is 50,000 km, 50,300 km on April 5, and 50,800 km on April 10 GPR would not simply draw straight lines between points or fit a fixed curve; instead, it predicts a smooth path based on the observed growth pattern, taking into account the "confidence" in each prediction. If April 5 to April 10 has fewer data points, GPR show a broader uncertainty band, reflecting less certainty in the predicted mileage.

Isotonic Regression fits a monotonically non-decreasing function to the data. Unlike other regression methods, it imposes a strict monotonicity constraint: the predicted values must never decrease as time progresses. It finds a piecewise-constant function that minimizes the total squared error under this constraint. The benefit of isotonic regression is that it is flexible for fitting non-parametric curves. This makes it highly suitable for modelling vehicle mileage which only increase over time. For example, if the vehicle's odometer shows 50,000 km on April 1, 50,200 km on April 6, and 50,400 km on April 10 isotonic regression fits a non-decreasing sequence that moves upwards from April 1 to April 10. Between April 1 and April 6, the estimated mileage stay flat for a few days and then increases sharply as new known readings are incorporated. This step-like behaviour ensures that the odometer never artificially dips, even when there are long gaps between observations.

After evaluating all four interpolation techniques (discussed in Section 5.1.1), isotonic regression is selected for the final system. In real vehicle data, isotonic regression produced a realistic mileage curve that grew consistently over time without introducing unrealistic decreases, making it the most reliable method for odometer interpolation among those evaluated. This method is applied to all vehicles with partial telemetry coverage, allowing accurate imputation of missing readings. The resulting interpolated values are integrated into the fault database, supporting downstream analysis and enabling accurate natural language querying through the QA pipeline.

4.3 Question Answering System Design

With the data in good shape, the next step is to build the QA system that interprets user queries and produces the corresponding SQL queries. This system went through iterations in development, particularly the prompting strategy. In the initial phase, open-source LLMs running locally, specifically CodeLLaMA and Mistral were experimented using the Ollama platform. This gave an insight into the capabilities and limitations of smaller-scale models for the NL2SQL task. Based on the challenges faced, the implementation was transitioned to using the Claude Application Programming Interface (API), which provides access to Anthropic's Claude, a state-of-the-art LLM.

This section describes the evolution of the QA system, including the reasoning behind moving to Claude, and details the final system's approach to prompt engineering, schema retrieval and dynamic prompt construction. The deployment of the QA system as an interactive chatbot using Streamlit is also discussed. Throughout the development, design decisions were consistently guided by the objective of maximizing accuracy and reliability in translating user questions into SQL queries, taking into account, the constraints and insights derived from experimental observations.

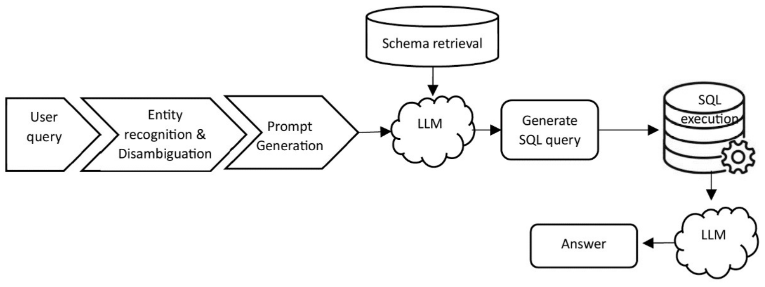


Figure 2. Schematic illustration of the question answering pipeline.

4.3.1 Initial Experiments with Local LLMs

The initial approach for building the QA system involved utilizing open-source LLMs that could run on local hardware, avoiding reliance on cloud APIs. CodeLLaMA 7B and Mistral 7B were selected as representative base models. CodeLLaMA, developed by Meta, is a variant of the LLaMA-2 family specifically fine-tuned for code generation tasks. Mistral 7B, developed by Mistral AI, is a newer model with 7 billion parameters that has been reported to outperform older, larger models, including LLaMA-2 13B, across several benchmarks. The motivation for exploring these models was to assess if a smaller, locally deployable model could handle the text-to-SQL conversion adequately. This would offer advantages in terms of data privacy by keeping everything on-premises and reducing costs.

Ollama which is used to manage the LLMs locally, simplified the process of loading the models. The prompt at this stage is relatively straightforward, typically instructing the model to convert a user question into an SQL query. Initial experiments also included adding schema context to the prompt, such as table and column names, to guide the model more accurately. A simplified example of such a prompt is shown in Figure 3. Complete examples of the prompts used are provided in Appendix A.

```
You are a database assistant.  Convert the user question into an SQL
query.  Use only the schema provided.

Table:  fault_events
Columns:
- [Column 1] (TYPE) - [Short description]
- [Column 2] (TYPE) - [Short description]
- ...

Instructions:

When generating SQL query, don't consider any column which is not
mentioned in the user query.
Never use JOIN on ''Main group'' or ''Sub Group Description''.  These are
columns, not separate tables.
Ensure column names are inside double quotes ("") for SQLite
compatibility.
Use GROUP BY for categorization and ORDER BY for sorting results.
```

Figure 3. Example of an early prompt given to local LLMs during initial experiments.

During these experimentations several challenges were encountered:

- *Hallucination:* The local models often hallucinated column names or table names that did not exist in the actual database. For example, given a question about engine faults, the model produces

  ```
  SELECT COUNT(*) FROM EngineFaults WHERE Year = 2024;
  ```

  even though no table called “EngineFaults” existed. The LLMs tend to fill in specifics even if not provided, and smaller models with limited training data might not reliably stick to the actual schema.

- *Logic mistakes:* Although both models are trained on general coding tasks, they are not specialized in SQL generation without domain-specific fine-tuning. For example, when asked to “list the top 5 vehicles with the highest number of faults”, the models generated incomplete queries, such as missing the GROUP BY clause. Without extensive prompt scaffolding, CodeLLaMA and Mistral frequently generated SQL with syntax or logic errors.

- *Knowledge Limitations:* Local models have a fixed amount of knowledge in their weights. This is evident when dealing with uncommon terms. For example, a query mentioning a specific component; e.g. ICL (Instrument Cluster), leads to an incorrect SQL query as it lacked contextual grounding for that term.

After these experiments, it became apparent that relying on local models would require significant prompt engineering and possibly fine-tuning to reach an acceptable level of accuracy. However, fine-tuning would require assembling a substantial domain-specific training set and was beyond the scope and timeline of this project. But the experiments were invaluable in highlighting the key needs: the model must be schema-aware to avoid hallucination, and it must be reliable in generating correct SQL syntax even if the question is complex. Given these insights, it was decided to shift to a more powerful model available via an API for the final system.

4.3.2 Transition to Claude API

While stronger models such as OpenAI's GPT-4 or SQL-specialized fine-tuned models exist, Claude was selected for both practical and technical reasons. Scania has an existing agreement with Amazon Bedrock and Anthropic for secure data transfer and API access, making Claude the most compliant and readily integrable option. Although fine-tuned models may outperform general-purpose LLMs on certain benchmarks, fine-tuning was not feasible in this project due to the absence of a sufficiently large domain-specific dataset. Claude has been trained with a focus on reliability and factuality through Anthropic's Constitutional AI alignment technique, which is designed to reduce the likelihood of generating incorrect or irrelevant SQL queries (Huang et al., 2023). Hence, Anthropic's Claude [2] accessed via Amazon bedrock was adopted to generate SQL queries. Claude was chosen for several reasons:

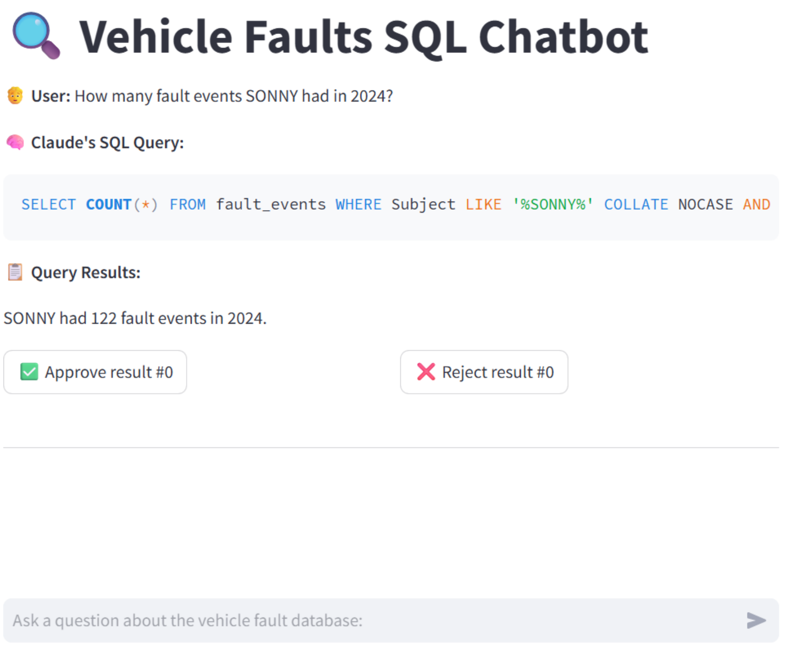
- *Superior Natural Language Understanding and Generation:* Claude has demonstrated excellent performance in understanding complex instructions and is less prone to errors. Its large model size and training on vast amounts of technical text and code make it much more reliable than small local models.
- *Better Adherence and Reasoning:* Claude's outputs were much more accurate on the first attempt. In cases involving ambiguity, it responded with clarifying questions, which indicates that it is actively "thinking" about the task. This reasoning ability was completely missing in smaller models.
- *Development Speed:* Switching to Claude eliminated the need for time-consuming fine-tuning. The Claude API is straightforward to integrate, enabling the back-end to send user queries as prompts and receive responses from the LLM within seconds.

In summary, the move to Claude addressed the earlier shortcomings. Hallucinations were virtually eliminated when the model was provided with actual schema context, and the ability to feed in more information made the system's prompt engineering task easier.

4.3.3 Streamlit-Based Chatbot Interface

The user interface is developed to facilitate interactive querying. Streamlit was selected for the front-end implementation, offering a lightweight framework to create a chatbot-like dialogue system. Users submit questions regarding vehicle faults, which trigger context retrieval, prompt construction and API interaction with Claude. The generated SQL query is executed against the database, and the result is returned to Claude to produce a natural language response. A *Human-in-the-Loop (HITL)* component is incorporated after the response is displayed, allowing users to evaluate the generated answer. Users are prompted to indicate whether the response is satisfactory or not, providing valuable feedback which is logged into a CSV file that records the original question, the generated SQL query, and the final response. The log is used to assess model performance and guide future improvements. This design aligns with best practices in human-AI collaboration, recognizing that even advanced LLMs like Claude may produce incorrect or suboptimal outputs (Huang et al., 2023). Figure 4 is a snapshot of the front-end user interface.

[2]https://www.anthropic.com/index/introducing-claude


Vehicle Faults SQL Chatbot
Approve result #0
Reject result #0
Ask a question about the vehicle fault database:

User: How many fault events SONNY had in 2024?

Claude's SQL Query:

```
SELECT COUNT(*) FROM fault_events WHERE Subject LIKE '%SONNY%' COLLATE NOCASE AND
```

Query Results:

SONNY had 122 fault events in 2024.

Figure 4. Chatbot interface.

4.3.4 Entity Recognition and Disambiguation

The first stage of the QA pipeline focuses on identifying and resolving entities in the user's query. *Named Entity Recognition (NER)* is implemented using the *spaCy* model (`en_core_web_sm`) [3] to tag standard entities such as dates, names, numbers, etc. In parallel, the domain-specific lexicon of unique values is used to resolve references to specific domain entities such as vehicle names, fault types, personnel names, and groups.

Following entity extraction, there is a disambiguation process to handle cases where a term could correspond to multiple columns in the schema. For example, the word "YENNEFER" could refer to a reporter name or a vehicle name, and the system would ask the user whether "YENNEFER" is meant as reporter or vehicle in that context. Such clarification dialogues ensure that the subsequent query uses the correct entity mapping and that multiple possible interpretations are resolved before execution.

To further improve the entity resolution, a fuzzy string matching using the *rapidfuzz* library is incorporated with a high similarity threshold to catch slight misspellings, abbreviations or partial matches of known values. If the similarity score between a token and a known lexicon entry exceeds the threshold of 85%, the token is treated as a match. This layered approach that combines NER, interactive disambiguation, and fuzzy lexical matching yields a precise mapping of query words to the database schema.

4.3.5 Prompt Generation

Prompt engineering refers to the process of designing structured inputs to guide the behaviour of LLMs. In this QA pipeline, prompt engineering plays a central role in translating user queries into accurate and executable SQL statements. Instead of providing the LLM with only the user's question, relevant context

[3]https://spacy.io/models/en#en_core_web_sm

from the underlying data is retrieved and prepended to the prompt. This technique, known as retrieval-augmented prompting, is supported by previous research showing that augmenting LLMs with external knowledge improves both the accuracy and factuality of their responses (Lewis et al., 2020). Here, the retrieval step uses the user's query to gather two types of context: schema context and domain context.

1. *Schema context retrieval:* To optimize query interpretation and SQL generation, a schema and value matching mechanism is employed to identify the database entities most relevant to a given user query. Lexical references such as 'mileage' or 'distance' are mapped to the odometer columns, whereas mentions of 'fault' or 'fault descriptions' are associated with the fault columns. This process utilizes a keyword matching technique, further enhanced through the integration of a domain-specific synonym set constructed from the curated knowledge base. In instances where the user query is broad or ambiguous, the full schema of database table is incorporated into the model prompt. Conversely, for targeted queries, the prompt is selectively constrained to only the relevant schema elements. This schema narrowing strategy serves to both reduce prompt complexity, thereby promoting model attentiveness, and to mitigate the risk of erroneous SQL generation involving unrelated database structures.
2. *Domain context retrieval:* To further enhance query understanding, domain-specific context is incorporated by retrieving relevant entries from the curated knowledge base. If the user query contains terms matching known values such as a vehicle name or a fault description, this information is provided to the language model. For example, if a query references "brake pressure low", the system supplements the prompt by indicating that it is a recognized fault description within the dataset, along with its corresponding fault category. A limited amount of representative data records is retrieved to illustrate the structure of the dataset to avoid overwhelming the model.

In this process, deterministic string matching operations are applied to collect relevant facts, which are incorporated into the input provided to the language model. This design is inspired from the RAG framework. By retrieving and presenting relevant schema information and known values at query time, the prompt effectively reminds Claude of the context, thereby improving the accuracy and relevance of the generated SQL queries. A full example of the structured prompt used is provided in Appendix A.

4.3.6 SQL Query Generation

SQL generation is the core step in which the user query is translated into a database query. Anthropic's Claude 3.5 Sonnet model is used to perform the translation. The prompt crafted with the database schema, relevant context, explicit instructions, and examples is provided to the LLM. The prompt also incorporates the results from the entity resolution. Additionally, a set of instructions is added within the prompt to steer the model's SQL generation. For instance, "Avoid using tables or columns that are not requested" prevents the model from making unwarranted assumptions. Certain guidelines on formatting like "Ensure column names are enclosed in double quotes (" ") for SQLite compatibility" is provided since the database column names have spaces. Another important instruction is to caution the model against using "JOIN" on descriptive fields, as these are often mistakenly treated as foreign keys or separate tables.

4.3.7 Function Calling

A function-calling style interface is employed to easily parse the output. The prompt defines a dummy function *generate_sql* and instructs Claude to respond with a call to that function, embedding the generated SQL query as an argument. This ensures that the output is returned as a well-structured JSON object and guarantees a clean SQL statement that can be executed on the database. To guide the model toward producing outputs in this format, few-shot example consisting of sample queries and SQL responses are included within the prompt. The final result is a clean, executable SQL query ready for direct execution.

```
{{
"reasoning": "<brief reasoning steps here>",
"function": "generate_sql",
"arguments": "<SQL query here>"
}}
```

Figure 5. Function calling

4.3.8 Query Execution

The next step is to execute the SQL query on the database. To protect the integrity of the data, a read-only format of the SQLite-based database is used to ensure that even if the model inadvertently generates a modification statement (e.g. DELETE, UPDATE), no data can be altered. Although prompt engineering and function-calling instructions constrain the model to generate SELECT-only queries, the read-only safeguard provides a secondary layer of protection. When a user submits a broad query such as “list all records” a result limit is also imposed to prevent unnecessary database load and reduce latency during large data transfers.

4.3.9 Answer Generation

Once the SQL query is executed, the results ranging from scalar values & single rows to larger results are post-processed into structured outputs. The result is returned to the model along with the user’s original query and a summarization instruction for larger results. To safeguard against hallucination, raw data is supplied with summarization prompt and instructions to restrict the model to factual inferences.

```
prompt = f“‘
You are an assistant that explains SQL query results concisely.

**User’s Question:**
user_query

**SQL Query Used:**
sql_query

**SQL Results:**
structured_results

Generate a clear, short, well-formatted, informative sentence that
clearly answers the user’s question based on this data.
Respond with a clear bullet-point format that groups faults by "Main
group", includes count if possible, and list each bullet point in a new
line.
’’’
```

Figure 6. Prompt template used to generate concise natural language summaries from SQL query results.

5 Evaluation & Discussion

The evaluation of the system is structured around the three research questions outlined in the introduction. The thesis aimed to assess the accuracy of LLMs in domain-specific NL2SQL tasks, the impact of data pre-processing and the role of prompting strategies in enhancing the quality of SQL queries generated by the LLM. To achieve these goals, quantitative accuracy assessments using a benchmark of domain-specific questions and qualitative analysis of various prompting techniques were employed. This section describes the evaluation methodology and results of the accuracy of NL2SQL, adequacy of data preparation and prompt design, as well as the discussion of the strengths, weaknesses, and lessons learned.

5.1 Data Quality Impact

A key aspect of the evaluation is determining how data preparation influenced system performance and correctness of the answers. The effectiveness of pre-processing steps is evaluated based on the consistency of the database and the system's ability to answer questions based on the cleaned data. The raw datasets initially contained various quality issues, and multiple data cleaning techniques such as deduplication, normalization, missing value imputation, and standardization were implemented. These are vital and the payoff is that the NL2SQL system operates on a coherent dataset.

5.1.1 Evaluation of Interpolation Models

One of the most significant pre-processing tasks is the interpolation of missing odometer readings. The fault logs recorded fault occurrences with time stamps, but not always the vehicle's mileage. Hence, for many fault entries, the exact odometer value is not recorded. However, knowing the odometer reading at the time of a fault is crucial to determine the vehicle's usage and fault severity. To fill this gap, four interpolation methods are experimented and performance compared.

The effectiveness of each method is compared on the basis of the following criteria: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), standard metrics for regression accuracy, and visual inspection for physical plausibility. The evaluation methodology involved creating synthetic missing values by randomly removing a subset (30%) of existing odometer readings for each vehicle. The models are then trained on the remaining data and tasked with predicting the artificially removed points. The predicted and true values are compared to compute RMSE and MAE. Lower RMSE and MAE indicates better model performance, implying that predictions are closer to actual outcomes.

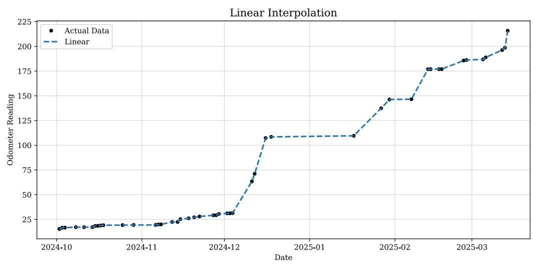
The average RMSE and MAE for all vehicles are summarized in Table 1.

Table 1

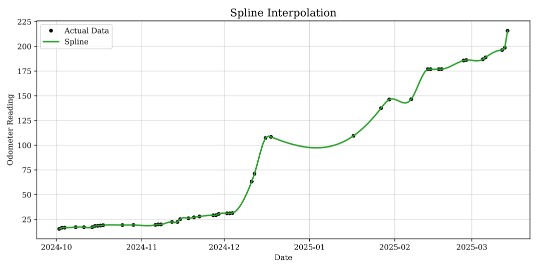
Performance comparison of interpolation methods across vehicles.

Model	**Avg. RMSE**	**Avg. MAE**
Linear Interpolation	22,073	16,349
Cubic Spline Interpolation	38,645,970	6,897,564
Gaussian Process Regression (GPR)	15,870	13,132
Isotonic Regression	16,103	13,321

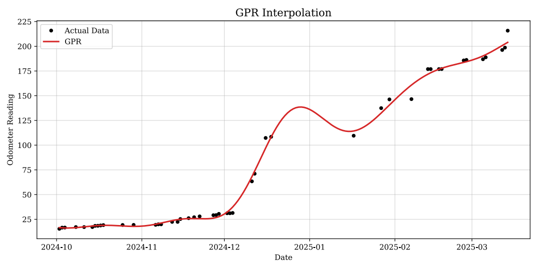
Linear interpolation performed acceptably, but is outperformed by GPR and isotonic regression in capturing non-linear trends. Spline interpolation yielded high errors and is ruled out. GPR achieved the lowest RMSE and MAE, while isotonic regression is close behind. However, numerical accuracy alone is insufficient for odometer modeling, where the underlying quantity should strictly increase over time. Upon further analysis, GPR showed minor dips in approximately 46.48% of the vehicle odometer trajectories evaluated. These dips are physically implausible (e.g., a vehicle appearing to lose kilometers). Given the importance of physical realism in mileage estimation, isotonic regression with strict monotonicity is ultimately selected for imputation in the final dataset.



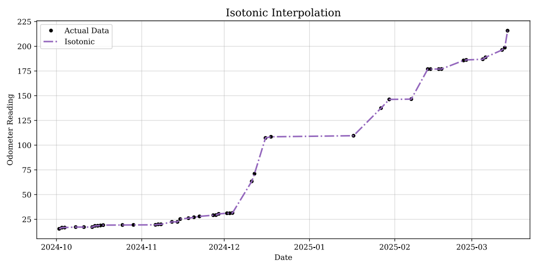
(a) Linear interpolation.



(b) Spline interpolation.



(c) GPR interpolation.



(d) Isotonic regression.

Figure 7. Comparison of interpolation methods for a vehicle.

Figure 7 shows the comparison of four interpolation methods applied to the odometer data for a vehicle. The actual recorded readings are shown as black dots, while the interpolated curves generated by Linear, Spline, GPR, and isotonic regression are overlaid. Each subplot illustrates how the different methods fits the odometer trajectory for the same vehicle, highlighting differences in smoothness, curvature, and monotonicity. Linear interpolation missed subtle curvature, leading to sharp transitions between recorded data points. Spline interpolation introduces a small dip around March–April 2024, indicating a temporary decrease in the odometer reading. GPR also exhibit a slight non-monotonic behaviour. Isotonic regression closely follows the actual odometer trajectory and shows the best overall agreement among the methods evaluated and is preferred for practical deployment.

5.1.2 Consistency & Reliability

After the pre-processing, the structured database is consistent and reliable. None of the errors during the QA evaluation could be attributed to data issues in which the LLM produced a correct SQL, but the answer is misleading due to data errors. Hence, RQ1 can be answered by stating that comprehensive data cleaning is indispensable for a robust natural language querying system.

5.2 NL2SQL Accuracy and LLM Performance

A benchmark set of 50 questions paired with an expected SQL query and the correct answer was constructed from the SQLite database. It covered queries involving event counts, attribute-based filtering, temporal conditions, and join operations. The correctness of the generated SQL is determined by two criteria: (1) string match against the expected SQL and (2) execution correctness. This ensures that the model not only generates syntactically correct queries but also semantically correct results.

The Claude Sonnet 3.5 model demonstrated robust natural language understanding through prompt engineering. Using the best prompt configurations (described in Section 5.3), it achieved 96% accuracy on the benchmark questions. The LLM's output is equivalent to the ground truth, though some SQL queries had interchangeable syntax that still yielded the correct answer. The achieved accuracy is due to the focused domain and the extensive prompt context provided, which significantly simplifies the task for the LLM.

5.2.1 Error Analysis

Although accuracy on the prepared queries is nearly perfect, it remains important to analyze the errors observed and potential errors that could occur in broader usage. Without schema guidance and lexicon, the model hallucinated column names and gave error messages. The implementation of fuzzy string matching against schema terms and values mitigated it by mapping terms to the correct database field. Additionally, the model was sensitive to unrecognized proper nouns or technical abbreviations. NER preprocessing using spaCy detected entities and cross-checked against the database lexicon, and if ambiguities or unknown entities are found, the system applied fuzzy matching or asked users for clarification before SQL generation.

5.2.2 Performance Analysis

CodeLLaMA and Mistral 7B achieved lower accuracy on the same set of queries, even when provided with schema context. This highlights that model capability and training play a critical role in the performance of NL2SQL systems. Fine-tuning smaller models may close the gap, but it is non-trivial without a significant training data. Instead, prompt-based utilization of a strong pretrained LLM like Claude 3.5 proves to be a practical and highly effective alternative. The accuracy results for all evaluated models are summarized in Table 2.

Table 2

Accuracy of different LLMs on the NL2SQL benchmark set.

Model	Accuracy (%)
CodeLLaMA 7B (local)	52
Mistral 7B (local)	56
Claude 3.5 Sonnet	96

These results affirm that, with the chosen Claude model and the selected approach, RQ2 is answered; the LLM can very effectively generate accurate SQL queries from natural language. The achieved accuracy demonstrates that LLMs when provided with schema context and domain knowledge, are capable of reliably bridging user questions to the underlying database. For a well-defined database, an LLM-powered solution can meet the accuracy standards to be practically useful.

5.3 Effect of Prompting

The third research question examines how different prompt engineering strategies affect the accuracy of LLM-generated SQL queries and the results derived from them. Throughout development, zero-shot prompting, schema-augmented prompts, instruction prompting, and few-shot examples were experimented with and evaluated to identify which approach yielded the best accuracy. Each approach is tested on the benchmark to determine the configuration that achieved the highest accuracy with minimal prompt length and latency. A zero-shot prompt enriched with schema context and explicit instructions provided the best results with fewer tokens.

Basic Zero-shot: The model is only given the user's question and a simple message like "You are an expert SQL assistant". This is the most minimal prompt among all prompt configurations. But without schema, the model often produced incorrect SQL queries. It had only an accuracy of 30% confirming that even powerful LLMs suffer from knowledge gaps if not grounded. For example "Count how many records in the fault log" produced *SELECT COUNT(*) FROM faults;* which looks correct, but "faults" table does not exist in the database.

Few-Shot Prompting: The model is provided with a few examples of question-to-SQL pairs in the prompt. With 4–5 examples, the model performed well on straightforward queries, but failed on complex or unseen queries. The added complexity of more example pairs make the prompt longer, increasing latency and cost per query.

Schema-Augmented Prompt: The prompt includes only a summary of the database schema. The accuracy increased to 80% on the test set. This proves that schema grounding is effective for LLM in performing the NL2SQL task more accurately in a domain-specific setting (X. Liu et al., 2024). The errors with this prompting were of a higher-level nature, like missing a filtering condition or misinterpreting the user's intent.

Schema + Instruction Prompt (Zero-shot): In addition to the schema, a detailed system instruction is included to guide the model on how to answer, handle uncertainties, and apply domain-specific reasoning. This configuration represents a zero-shot with schema and a task-specific context approach. For instance, when asked "How many occurrences of fault "X" in January?" the model responded with "Do you mean January of any specific year? And for all vehicles or a particular one?" demonstrating its ability to proactively seek clarification. This behaviour improves the quality of user interaction and contributes to greater trust in the responses.

CoT + Schema + Instruction Prompt (Few-shot): The model is instructed to articulate its reasoning step by step before generating SQL queries. Initially, the CoT prompt produced inconsistent outputs when used without few-shot examples. However, after incorporating schema context, task-specific instructions, and a set of examples, the approach yielded significantly improved results, achieving 96% accuracy on the benchmark queries. The step-by-step reasoning provided transparency and trust, but latency increased slightly due to the additional reasoning step.

Table 3

Accuracy of different prompting strategies on benchmark dataset

Prompting Strategy	**Accuracy (%)**
Basic Zero-shot	30
Few-shot Prompting	60
Schema-Augmented Prompt	80
Schema + Instruction Prompt (Zero-shot)	96
CoT + Schema + Instruction (Few-shot)	96

As shown in Table 3, the accuracy of LLM-generated SQL queries improved significantly with better prompt design. This addresses RQ3 by demonstrating that prompt strategies significantly influence both the accuracy and capability of LLM to handle domain-specific queries. Although both the schema + instruction (zero-shot) and CoT based few-shot strategies achieved the same accuracy (96%), the few-shot approach resulted in longer prompts and slightly higher latency without improving accuracy. Therefore, the zero-shot schema + instruction configuration is considered more efficient and practical for this domain.

5.4 Discussion

The evaluation provides clear insights into the factors that influence the effectiveness of LLMs in domain-specific NL2SQL tasks. The findings from each research question collectively highlight the critical roles of data preparation, prompt engineering, and model capability in achieving high accuracy and reliability in natural language interfaces over structured databases.

In addressing RQ1, it became evident that data quality has a major impact on the system's performance. The original dataset contained various inconsistencies and missing values, particularly in odometer readings, which are crucial for interpreting fault severity and vehicle usage patterns. Among the interpolation techniques tested, GPR achieved the best numerical accuracy in terms of RMSE and MAE. However, it violated physical constraints by producing decreasing mileage trajectories. In contrast, isotonic regression, although slightly less precise numerically, preserved the monotonicity expected for cumulative distance measures. This trade-off illustrates that in engineering applications, domain-valid assumptions can be more important than optimizing statistical metrics alone. The resulting pre-processed dataset was coherent and reliable, providing a strong foundation for accurate question answering and validating the importance of thorough data preparation.

In response to RQ2, the Claude Sonnet 3.5 model demonstrated strong performance, achieving 96% accuracy on the benchmark queries. The accuracy assessment considered both syntactic correctness of the generated SQL queries and semantic alignment with the expected results. Most errors occurred when schema information or domain-specific terms are missing or ambiguous. These were effectively mitigated by integrating NER to detect key tokens, fuzzy matching against known schema elements, and a domain lexicon to resolve ambiguity. The integration of NLP techniques with modern LLMs significantly enhanced the robustness of the system, demonstrating that LLMs alone are not sufficient: they require structured guidance and validation mechanisms to perform reliably in constrained real-world environments.

For RQ3, the evaluation revealed that prompt engineering plays a critical role in NL2SQL performance. Starting from basic zero-shot prompting with minimal context, which achieved only 30% accuracy, each enhancement, such as schema grounding and explicit task instructions, led to substantial improvements. The most effective configuration is the zero-shot prompt augmented with schema context and clear instructions, reaching 96% accuracy. While CoT prompting introduced intermediate reasoning steps that improved interpretability, it did not yield higher accuracy in this limited domain and slightly increased prompt length and latency. Nevertheless, in more complex or broader domains, CoT prompting may provide additional benefits, especially for handling compositional or multi-step reasoning tasks. In this feasibility study, the simpler schema + instruction prompt proved sufficient, but CoT remains a promising approach for future scalability and robustness.

The human-in-the-loop component plays a supportive role in the evaluation of the system. Although feedback is not used for detailed error analysis or model tuning, it can provide a rich dataset for supervised fine-tuning or training a feedback-aware ranking module. Evaluation metrics such as BLEU or ROUGE are not used in this study, as they are generally not suited for NL2SQL tasks where multiple query formulations may produce the same result. Instead, the evaluation focuses on execution correctness and string match against reference queries.

Overall, the evaluation reveals that effective NL2SQL systems depend on the synergy between a well-prepared data, strategically engineered prompts, and capable language models. This thesis demonstrates that with the right combination of preprocessing, prompt design, and model selection, it is possible to build a reliable and efficient natural language interface for querying structured databases in industrial applications. The design of showing the generated SQL to the user and the use of a read-only database connection are strengths in terms of transparency and safety. The system's response times were generally in the order of 5–6 seconds for a query which is acceptable for interactive use.

6 Conclusion

This thesis presents a question-answering interface that enables users to interact with a structured vehicle fault log database using natural language, powered by large language models. The core contribution is a pipeline that combines carefully prepared data with advanced prompt engineering strategies to deliver accurate and user-friendly NL2SQL performance in a real-world industrial setting.

Evaluation results demonstrate that accurate SQL queries can be generated from natural language with near-perfect accuracy, validating the effectiveness of the approach. This is achieved without task-specific training data, highlighting that a large pre-trained model can be successfully adapted to specialized domains through contextual prompting alone. The findings also underscore the importance of clean structured data in eliminating potential errors.

Furthermore, isotonic regression is identified as a promising method for predicting cumulative time-series metrics such as odometer readings, counters, or progress indicators. Both the schema + instruction (zero-shot) prompt and the chain-of-thought (few-shot) prompt achieved 96% accuracy on domain-specific queries, affirming the value of prompt engineering for guiding model behaviour.

Overall, this work serves as a case study in bridging the gap between advanced AI capabilitiess and practical industry needs. By removing the technical barrier of SQL knowledge, the proposed solution enables faster troubleshooting, more efficient data-driven decision making, and improved communication between technical and non-technical stakeholders.

6.1 Limitations

Despite its successes, this work has certain limitations that must be acknowledged. The solution is developed and tested for a specific use case (Scania's vehicle fault logs). As a result certain design choices like prompt phrasing and interpolation model parameters are tuned to this scenario. The underlying database used consists of only a few thousand records and features a relatively simple relational structure. If attempted to apply the same LLM prompt to a massive enterprise data warehouse with numerous interconnected tables, new challenges would emerge.

Moreover, the current implementation is not designed for real-time query scenarios or is integrated with live systems that might want to query continuously updating data. A production-grade deployment would necessitate additional engineering efforts to manage periodic data refresh, cache frequent queries, and maintain system responsiveness. Furthermore, no mechanism was implemented to enable learning from user interactions over time and incorrect responses are not used to improve future performance. Incorporating such feedback loops could significantly enhance adaptability and robustness in a dynamic environment.

6.2 Future Work

The results of this thesis open several promising directions for future work, both to address current limitations and to extend overall capability. While the approach demonstrated high accuracy on the benchmark queries, the size of the evaluation set remains limited. Future work could involve a more comprehensive evaluation using a larger and more diverse set of questions to better assess generalizability and reliability.

A next step is to enable more open-ended forms of analysis, like integrating a statistical module or leveraging the LLM's ability to perform reasoning over retrieved data. One possible approach could be to have the LLM generate a small script to perform more complex analysis. For instance, the model generates SQL and then Python code to visualize the results using tools like Jupyter-like computational back-ends. Such capabilities would significantly expand the interface's utility for exploratory data analysis and bring it closer to functioning as a true data science assistant.

A more ambitious direction for future research is to explore agentic behaviour in LLMs over structured databases. Rather than generating a single SQL query in response to each question, the model could be prompted to outline a reasoning plan, run several queries to explore different aspects, and then consolidate the results. This approach shifts the role of the LLM from a query generator to an intelligent autonomous planner and executor that reason over databases.

Another direction is replacing or supplementing Claude 3.5 with an open-source LLM fine-tuned for SQL generation to reduce dependency on external services. The NL-SQL pairs can be collected and saved from the current usage to fine-tune a smaller model. Although it may initially lag in performance, the gap between closed and open models continue to narrow. This would allow on-premise deployment, important for data privacy, cost reduction, and faster response times.

In summary, this thesis lays a solid foundation for natural language querying over structured technical data. When guided by clean data and carefully engineered prompts, LLMs prove to be highly effective in translating user queries into executable database commands. Continued advances in model architecture, prompt strategies, and feedback integration promise to further enhance such systems, making them increasingly viable for complex and large-scale industrial applications. The findings and methods from this work are likely to inspire future research in building intelligent, accessible interfaces for structured data exploration.

References

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

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., ... Kaplan, J. (2022). Constitutional AI: harmlessness from AI feedback. *CoRR*, *abs/2212.08073*. https://doi.org/10.48550/ARXIV.2212.08073

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Date, C. J. (2000). *An introduction to database systems (7. ed.)* Addison-Wesley-Longman.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 4171–4186. https://doi.org/10.18653/V1/N19-1423

Dong, L., & Lapata, M. (2018). Coarse-to-fine decoding for neural semantic parsing. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, 731–742. https://doi.org/10.18653/V1/P18-1068

Eyal, B., Mahabi, M., Haroche, O., Bachar, A., & Elhadad, M. (2023). Semantic decomposition of question and SQL for text-to-sql parsing. *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, 13629–13645. https://doi.org/10.18653/V1/2023.FINDINGS-EMNLP.910

Finegan-Dollak, C., Kummerfeld, J. K., Zhang, L., Ramanathan, K., Sadasivam, S., Zhang, R., & Radev, D. R. (2018). Improving text-to-sql evaluation methodology. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, 351–360. https://doi.org/10.18653/V1/P18-1033

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., & Zhou, J. (2024). Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, *17*(5), 1132–1145. https://doi.org/10.14778/3641204.3641221

Ghosal, D., Majumder, N., Poria, S., Chhaya, N., & Gelbukh, A. F. (2019). Dialoguegcn: A graph convolutional neural network for emotion recognition in conversation. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, 154–164. https://doi.org/10.18653/V1/D19-1015

Groff, J. R., & Weinberg, P. N. (2002). *Sql: The complete reference*. McGraw-Hill/Osborne Media.

Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J., Liu, T., & Zhang, D. (2019). Towards complex text-to-sql in cross-domain database with intermediate representation. *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, 4524–4535. https://doi.org/10.18653/V1/P19-1444

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., & Liu, T. (2023). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *CoRR*, *abs/2311.05232*. https://doi.org/10.48550/ARXIV.2311.05232

Inui, K., Jiang, J., Ng, V., & Wan, X. (Eds.). (2019). *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing, EMNLP-IJCNLP 2019, hong kong, china, november 3-7, 2019*. Association for Computational Linguistics. https://aclanthology.org/volumes/D19-1/

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de Las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., & Sayed, W. E. (2023). Mistral 7b. *CoRR*, *abs/2310.06825*. https://doi.org/10.48550/ARXIV.2310.06825

Kim, H., So, B., Han, W., & Lee, H. (2020). Natural language to SQL: where are we today? *Proc. VLDB Endow.*, *13*(10), 1737–1750. https://doi.org/10.14778/3401960.3401970

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

Li, F., & Jagadish, H. V. (2014). Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, *8*(1), 73–84. https://doi.org/10.14778/2735461.2735468

Lin, X. V., Socher, R., & Xiong, C. (2020). Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, 4870–4888. https://doi.org/10.18653/V1/2020.FINDINGS-EMNLP.438

Liu, G., Tan, Y., Zhong, R., Xie, Y., Zhao, L., Wang, Q., Hu, B., & Li, Z. (2025). Solid-sql: Enhanced schema-linking based in-context learning for robust text-to-sql. *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, 9793–9803. https://aclanthology.org/2025.coling-main.654/

Liu, X., Shen, S., Li, B., Ma, P., Jiang, R., Luo, Y., Zhang, Y., Fan, J., Li, G., & Tang, N. (2024). A survey of NL2SQL with large language models: Where are we, and where are we going? *CoRR*, *abs/2408.05109*. https://doi.org/10.48550/ARXIV.2408.05109

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, *abs/1907.11692*. http://arxiv.org/abs/1907.11692

Melton, J., & Simon, A. R. (1993). *Understanding the new SQL: A complete guide*. Morgan Kaufmann.

Nan, L., Zhao, Y., Zou, W., Ri, N., Tae, J., Zhang, E., Cohan, A., & Radev, D. (2023). Enhancing text-to-sql capabilities of large language models: A study on prompt design strategies.

Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023, 14935–14956. https://doi.org/10.18653/V1/2023.FINDINGS-EMNLP.996

Pourreza, M., & Rafiei, D. (2023). DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. http://papers.nips.cc/paper%5C_files/paper/2023/hash/72223cc66f63ca1aa59edaec1b3670e6-Abstract-Conference.html

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training [Technical report]. *OpenAI Blog*. https://openai.com/research/language-unsupervised

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, *21*, 140:1–140:67. https://jmlr.org/papers/v21/20-074.html

Rajkumar, N., Li, R., & Bahdanau, D. (2022). Evaluating the text-to-sql capabilities of large language models. *CoRR*, *abs/2204.00498*. https://doi.org/10.48550/ARXIV.2204.00498

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. MIT Press. https://www.worldcat.org/oclc/61285753

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Canton-Ferrer, C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., … Synnaeve, G. (2023). Code llama: Open foundation models for code. *CoRR*, *abs/2308.12950*. https://doi.org/10.48550/ARXIV.2308.12950

Rubin, O., Herzig, J., & Berant, J. (2022). Learning to retrieve prompts for in-context learning. *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, 2655–2671. https://doi.org/10.18653/V1/2022.NAACL-MAIN.191

Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. *CoRR*, *abs/2402.07927*. https://doi.org/10.48550/ARXIV.2402.07927

Scholak, T., Schucher, N., & Bahdanau, D. (2021). PICARD: parsing incrementally for constrained auto-regressive decoding from language models. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, 9895–9901. https://doi.org/10.18653/V1/2021.EMNLP-MAIN.779

Sun, R., Arik, S. Ö., Nakhost, H., Dai, H., Sinha, R., Yin, P., & Pfister, T. (2023). Sql-palm: Improved large language model adaptation for text-to-sql. *CoRR*, *abs/2306.00739*. https://doi.org/10.48550/ARXIV.2306.00739

Sütőová, A. (2018). Improving information flow for decision making on product quality in the automotive industry. *Quality Innovation Prosperity*, *22*(1), 73–80.

Tai, C., Chen, Z., Zhang, T., Deng, X., & Sun, H. (2023). Exploring chain of thought style prompting for text-to-sql. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing,* *EMNLP 2023, Singapore, December 6-10, 2023*, 5376–5393. https://doi.org/10.18653/V1/2023.EMNLP-MAIN.327

Tan, Z., Liu, X., Shu, Q., Li, X., Wan, C., Liu, D., Wan, Q., & Liao, G. (2024). Enhancing text-to-sql capabilities of large language models through tailored promptings. *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, 6091–6109. https://aclanthology.org/2024.lrec-main.539

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). Llama: Open and efficient foundation language models. *CoRR*, *abs/2302.13971*. https://doi.org/10.48550/ARXIV.2302.13971

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 5998–6008.

Wang, B., Shin, R., Liu, X., Polozov, O., & Richardson, M. (2020). RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, 7567–7578. https://doi.org/10.18653/V1/2020.ACL-MAIN.677

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. http://papers.nips.cc/paper%5C_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

Yavuz, S., Gur, I., Su, Y., & Yan, X. (2018). What it takes to achieve 100 percent condition accuracy on wikisql. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 1702–1711. https://doi.org/10.18653/V1/D18-1197

Yin, P., Neubig, G., Yih, W., & Riedel, S. (2020). Tabert: Pretraining for joint understanding of textual and tabular data. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, 8413–8426. https://doi.org/10.18653/V1/2020.ACL-MAIN.745

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., & Radev, D. R. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 3911–3921. https://doi.org/10.18653/V1/D18-1425

Zelle, J. M., & Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2*, 1050–1055. http://www.aaai.org/Library/AAAI/1996/aaai96-156.php

Zhang, Y., Radishian, C., Brunswicker, S., Whitenack, D., & Linna, D. W. (2024). Empathetic language in llms under prompt engineering: A comparative study in the legal field. *6th International Conference on AI in Computational Linguistics, ACLING 2024, September 21-22, 2024, Dubai, UAE*, 308–317. https://doi.org/10.1016/J.PROCS.2024.10.204

Zhao, X., Zhou, X., & Li, G. (2024). Chat2data: An interactive data analysis system with rag, vector databases and llms. *Proc. VLDB Endow.*, *17*(12), 4481–4484. https://doi.org/10.14778/3685800.3685905

Zhong, V., Xiong, C., & Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, *abs/1709.00103*. http://arxiv.org/abs/1709.00103

Ziletti, A., & D'Ambrosi, L. (2024). Retrieval augmented text-to-sql generation for epidemiological question answering using electronic health records. *Proceedings of the 6th Clinical Natural Language Processing Workshop, ClinicalNLP@NAACL 2024, Mexico City, Mexico, June 21, 2024*, 47–53. https://doi.org/10.18653/V1/2024.CLINICALNLP-1.4

Appendices

Appendix A Prompts

This appendix includes examples of prompts used during the development and evaluation of the system. These prompts were dynamically constructed at runtime and passed to the LLM to generate SQL queries.

A.1 Basic Zero-shot Prompt

```
You are an expert SQL assistant. Convert the user question into a valid SQL
query.
```

A.2 Few-shot Prompt

```
You are an expert SQL assistant. Convert the user question into a valid SQL
query using only the table and columns listed.

Table: ‘‘fault_events’’
Columns: Column1, Column2,....

Example Queries and Correct SQL Outputs:
    - ‘‘How many TPM faults were reported by BLUES in 2025?’’
       ´´´sql
       SELECT COUNT(*)
       FROM ‘‘fault_events’’
       WHERE STRFTIME(‘%Y’, ‘‘Date of failure‘‘) = ‘2025’
       AND LOWER(‘‘Subject’’) = ‘blues’
       AND (
           LOWER(‘‘Main group’’) LIKE ‘%tpm%’
           OR LOWER(‘‘Sub Group Description’’) LIKE ‘%tpm%’
       );

    - ‘‘What type of faults has SONNY had?’’
       ´´´sql
       SELECT DISTINCT ‘‘Main group’’, ‘‘Sub Group Description’’
       FROM ‘‘fault_events’’
       WHERE ‘‘Subject’’ = ‘SONNY’
       ORDER BY ‘‘Main group’’ ASC, ‘‘Sub Group Description’’ ASC;

....
```

A.3 Schema-Augmented Prompt

```
You are an expert SQL assistant. Convert the user question into a valid SQL
query using only the table and columns listed.

Table: ‘‘fault_events’’
Columns:
```

```
[‘‘Column 1’’] (TYPE) -- [Short description]
[‘‘Column 2’’] (TYPE) -- [Short description]
...
```

A.4 Schema + Instruction Prompt (Zero-shot)

```
You are an expert SQL assistant. Convert the user question into a valid SQL
query using only the table and columns listed.

Table: ‘‘fault_events’’
Columns:
[‘‘Column 1’’] (TYPE) -- [Short description]
[‘‘Column 2’’] (TYPE) -- [Short description]
...

Instructions:
- When generating SQL query, don’t consider any column which is not mentioned
  in the user query.
- Do NOT include ‘‘Status’’ in the WHERE clause unless the user explicitly asks
  for it.
- If searching faults, always check in ‘LOWER(‘‘Main group’’)’ or
  ‘LOWER(‘‘Sub Group Description’’)’ using LIKE for partial matches.
- Use ‘LOWER(...)’ LIKE ‘%keyword%’ for text searches.
- Use inferred column and exact value match if user input matches known label.
- When answering about odometer values, always check both ‘‘Actual odometer’’
  and ‘‘Predicted odometer’’.
...
```

A.5 CoT + Schema + Instruction Prompt (Few-shot)

```
You are an expert SQL assistant. Convert the user question into a valid SQL
query using only the table and columns listed.

Table: ‘‘fault_events’’
Columns:
[‘‘Column 1’’] (TYPE) -- [Short description]
[‘‘Column 2’’] (TYPE) -- [Short description]
...

Instructions:
- Always explicitly include brief reasoning steps in your response.
- Always perform the following steps explicitly:
    1. Think step-by-step about the user’s natural language query.
    2. Identify relevant database columns based on the provided schema.
    3. Clearly outline your reasoning (briefly) before generating SQL.
    4. Generate accurate SQL queries based on your reasoning.
- For odometer readings, use ‘COALESCE(‘‘Actual odometer’’, ‘‘Predicted
  odometer’’)’.
- Use LIKE and COLLATE NOCASE for case-insensitive partial matching.
...
```

```
Example output:
    {{
    ‘‘reasoning’’: ‘‘User asked about the number of faults reported by BLUES.
    The relevant column is ‘Reporter’. Counting all entries where
    Reporter=‘BLUES’.’’,
    ‘‘function’’: ‘‘generate_sql’’,
    ‘‘arguments’’: ‘‘SELECT COUNT(*) FROM fault_events WHERE Reporter LIKE
    ‘%BLUES%’ COLLATE NOCASE;’’
    }}
```

Appendix B Background on Scania AB

Scania AB is a world-leading manufacturer of heavy commercial vehicles, engines, and services, headquartered in Södertälje, Sweden. Founded in 1891, the company has grown into a global brand with operations in more than 100 countries. Scania is part of the TRATON GROUP, a subsidiary of the Volkswagen Group.

The company specializes in the production of trucks and buses for heavy transport applications, as well as industrial and marine engines. In addition to manufacturing, Scania offers a wide range of connected services that support fleet management, sustainability, and operational efficiency. These services are enabled by Scania’s commitment to digitalization and data-driven innovation.

At the core of Scania’s development process is a strong focus on vehicle testing and validation. Extensive test fleets generate large volumes of structured and unstructured data, including fault logs, sensor data, and driver feedback. Analyzing this data is critical for quality assurance, product development, and pre-release reliability assessments.

In recent years, Scania has invested heavily in digital transformation, including advanced analytics, artificial intelligence, and cloud infrastructure. These technologies support more intelligent product development cycles, predictive maintenance, and smarter transportation solutions.

This thesis was conducted in collaboration with Scania’s Research and Development division and focuses on building a natural language interface to query structured fault log data. This effort aligns with Scania’s broader goals of increasing data accessibility, supporting non-technical users, and accelerating data-informed decision-making within engineering workflows.