CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Breaking The Token Barrier: Leveraging Retrieval Methods to Facilitate Text-to-SQL on Extensive Tabular Data

Master's thesis in Computer science and engineering

VICTOR BERGSTRÖM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Breaking The Token Barrier: Leveraging Retrieval Methods to Facilitate Text-to-SQL on Extensive Tabular Data

VICTOR BERGSTRÖM

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Leveraging Retrieval Methods to Facilitate Text-to-SQL on Extensive Tabular Data

VICTOR BERGSTRÖM

Supervisor: Miroslaw Staron, Computer Science and Engineering
Advisor: Yudi Chen, Zenseact
Examiner: Mats Granath, Department of Physics

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Leveraging Retrieval Methods to Facilitate Text-to-SQL on Extensive Tabular Data

VICTOR BERGSTRÖM
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Technical analysis of systems is extensively utilized in the industrial sector, particularly in automotive development. This analysis is often conducted on logs generated during a system's runtime, which typically contain sensor values, timestamps, and events. These logs represent the primary means for analyzing a system's behavior and identifying failures. But, depending on their size and structure, finding the correct data in these tables can be difficult. At Zenseact, their large table with sensor data remains largely unused by developers due to the challenges associated with acquiring sufficient knowledge to leverage it effectively. Full utilization of this data could speed up development and allow for more robust software to be developed by enabling an easier way to find the cause of errors. This thesis suggests leveraging LLMs to perform text-to-SQL to interface with this table. Here, a developer could use natural language and the LLM would generate the SQL code and query the database. But current text-to-SQL methods are naive, they assume the table's schemas are small enough to fit inside the LLM's token limit which is not always the case, especially for Zenseact's table with 5677 columns. This thesis tackles the overarching problem by crafting an artifact that employs retrieval methods to fetch only the related columns of the schema given a user's natural language query. If this works, it also has the added benefit of simplifying the problem for the LLM where the columns it is presented with are fewer and relevant to the query. The performance is measured in Errorless Code ratio. This ratio measures how often the generated SQL queries can run, without errors, on the database. This does not take the correctness of the code into account. The results were measured using 0, 1, and 2 shot and came out to 35%, 34%, and 39% respectively. To improve performance, the problem was divided by letting the LLM in two separate steps, first select columns and then generate the SQL code. This approach improved the EC to 57%.

The retrieval was benchmarked separately using 18 tests. Default retrieval, where the column is embedded and stored in a vector database yielded a result of 4 out of 18 on the benchmark. To improve retrieval performance the columns were simplified using a term frequency removal algorithm. This improved the result to 6 out of 18 correct on the retrieval benchmark.

The artifact presents reasonable approaches for text-to-SQL. Retrieval is a well-established method in research, and the 2-shot and refined column selection both improve a text-to-SQL system's performance. However, using systems like this in real scenarios introduces a big hurdle for the models used. It's almost impossible for the model to have learned enough in its initial training to understand the company-specific information. The methods presented for text-to-SQL on extensive database tables could be viable, but further improvements to the AI models are required until

they can be used in a production environment.

Keywords: Computer, science, computer science, engineering, project, thesis.

Acknowledgements

I am grateful to Zenseact for their support during my project and providing me with the necessary resources to conduct my research. Special thanks to my supervisors at Zenseact Yudi Chen and Yuchuan Jin, and my Chalmers supervisor Miroslaw Staron. I also want to thank my colleagues Jesper Knapp and Klas Moberg for their camaraderie and valuable insights.
These peoples' support has been invaluable in shaping this thesis.

Victor Bergström, Gothenburg, 2024-06-07

Contents

List of Figures

List of Tables

1

Introduction

Technical analysis of systems is extensively utilized in the industrial sector, particularly in automotive development, to analyze the causes of failures and prevent them to improve system dependability. This analysis is often conducted on logs generated during a system's runtime, which typically contain sensor values, timestamps, and events. These logs represent the primary means for analyzing a system's behavior and identifying failures[1].

At Zenseact, hundreds of thousands of logs are generated from different teams and test systems every day. Billions of logs accumulate each year, and manually searching through these logs has proven impractical. Part of these logs consists of the **vehicle logs** generated from test drives with Zenseacts automotive cars. The logs contain a table with 5,677 columns of sensor data. What can be done to enable developers to effectively search through these logs? When a developer searches this table, they have to develop Structured Query Language (SQL) queries. The problem posed by this is that to effectively use this table, not only does the developer need to know SQL, but knowledge of each column is necessary, and learning over 5,000 sensor values is not feasible. A much more effective approach would be for a developer to describe a scenario they are looking for in natural language, and some system then translates this to SQL code and queries the table.

Converting natural language to code aims to enhance a developer's efficiency and speed. While large language models (LLMs) such as GPT-4 [2] excel at this task, these large models are often too large to run on available hardware efficiently. GPT-4s predecessor GPT-3.5 has around 175 billion parameters[3]. Recently, smaller specialized LLMs boast great performance in tasks such as coding [4] [5]. With performance comparable to their much larger siblings, but significantly less computationally intensive. Deepseek-coder with its 6.7 billion parameters is one of these smaller models [6] consistently scoring high on coding leaderboards like CanAiCode and EvalPlus. A further fine-tuned version of Deepseek-coder, Natural-SQL has been trained on SQL tasks, achieving a respectable score of 76.5% on Defog's SQL-eval vs. GPT-4's 83% and GPT-3.5's 65% [7][8].

LLMs utilizing transformers have an inherent maximum context window, limiting the amount of information the LLM can utilize in a single conversation [7][9]. This causes an issue for text-to-SQL when the database schema, such as Zenseact sensor data, exceeds this context. A solution to this is to provide only the information rel-

evant to each user query to the LLM [10]. This can be done using retrieval methods presented in this thesis.

This work proposes a system to enhance the user experience when working with large databases. By leveraging LLMs, can we remove the prerequisite knowledge, enabling developers to query Zenseact's large database effectively? In this thesis, a framework for utilizing LLMs is employed to facilitate SQL code generation on large SQL tables.

1.1 Purpose of the study

At Zenseact, the large table with sensor data remains largely unused by developers due to the challenges associated with acquiring sufficient knowledge to leverage it effectively. Full utilization of this data could speed up development and allow for a more robust software to be developed by enabling an easier way to find the cause of errors. But, because of these aforementioned prerequisites stakeholders and developers at Zenseact can only guess at its value. This study aims to devise a novel solution to tackle the obstacles presented by large tabular datasets, with a specific focus on Zenseact's extensive sensor data, within the domain of employing modern LLMs for text-to-SQL conversion. The primary objective is to improve the efficiency and efficacy of querying and analyzing significantly large tabular datasets, particularly in instances where the dataset surpasses the token/context limitations imposed by LLMs. This thesis will assess the text-to-SQL performance of an artifact crafted using existing AI models, and thereby discerning their potential in the realm of text-to-SQL.

1.2 Research Questions

This thesis investigates solutions within the domain of text-to-SQL, focusing on the framework for converting natural language prompts into SQL code and assessing its efficacy. The main focus will be:

- **RQ 1:** *To which degree can retrieval methods select appropriate columns in a large SQL Schema to allow for text-to-SQL to be performed?* By addressing this question, we can determine whether retrieval is a viable method for retrieving the necessary information on a per-user query basis, enabling the LLM to solve text-SQL tasks on large SQL schemas.

Xue-Yong et al[11] write about the difficulties associated with the deployment of LLMs in real-world settings, particularly emphasizing the constraints posed by hardware limitations, cost, and closed API cost considerations. Most text-to-SQL solutions presented in BIRD, and SPIDER use LLMs with parameters above 100 billion [12][13] which can't run on most systems. This project emphasizes using typical open fine tuned models. The common fine tuned model is available in different sizes but models with around 7 billion parameters are common[14][6][7].

- **RQ 2:** *To what extent can current open and deployable 7B LLMs work for Text To SQL tasks?* This will determine what performance can be expected when deploying an open and available LLM on local system.

1.3 Limitations

Infrastructure Setup: The initial phase of this thesis involves development for hosting the in-house language models. The deployment of and the hosting of an API on the virtual machine entails a time-consuming development process. The allocation of time for this infrastructure may divert attention from core activities, potentially impacting the overall timeline and focus of the study.

Performance Constraints: The provided resources limit the model candidates, the models must fit within the memory on the available GPU. Fine-tuning large language models is also highly resource-intensive, as a result, using fine-tuning to improve text-to-SQL will be avoided in favor of Few-shot-prompting.

Limited Generalization: The study's applicability is limited by the specific focus on logs generated from Zenseacts test vehicles. While the intention is to develop a specialized system for parsing vehicular data, the generalization of findings to other domains may be limited, necessitating extra development in incorporating the results beyond the scope of the targeted datasets.

2

Background

This chapter will delve into the fundamental concepts in Natural Language Processing (NLP) and Large Language Models (LLM). First, it introduces tokens and embeddings which are the first steps in how retrieval and LLMs work. The following sections describe the workings of LLMs and retrieval systems. Finally, the chapter examines current literature on text-to-SQL and how their models work.

2.1 Tokenization & Embeddings

When using neural networks the input and output are typically made up of vectors or matrices containing numbers. This means, in order for a model to work with natural language we first need to convert text into some numerical representation.

First, we introduce tokenization, this is a fundamental preprocessing step in NLP that involves breaking down text into smaller units called tokens. These tokens can be words, subwords, or characters, depending on the desired granularity. Tokenization methods vary depending on the specific task and requirements. A common technique is to break down words into smaller units to handle variations and out-of-vocabulary words effectively. Words along with commonly occurring strings of characters such as "walk", "ing", or "ed" are converted into a numerical representation. An example is shown in figure 2.1. This tokenizer is just an illustrative example, and there are many different ways to tokenize. OpenAI uses tiktoken, a version of BPE [15], other examples of tokenizers are WordPiece and SentencePiece [16].

Tokenizers work well for extracting features from text, but the tokens themselves don't contain any actual meaning. The solution for this is to employ embeddings where each token gets assigned a unique point in a high-dimensional space. In this space similar tokens are closer together as shown in 2.2. This allows the neural network to understand relationships between words and their meanings. Mikolov et al. [17] introduced word2vec, which captures semantic relationships between words in a 600-dimensional vector. Their model learns distributed representations of words, by training a neural network on large text corpora. Through this process, words with similar contexts are positioned closer together in the embedding space, facilitating the model's ability to capture semantic similarities. The first layers in LLMs are often encoder modules that embed the input token before the calculations are performed.

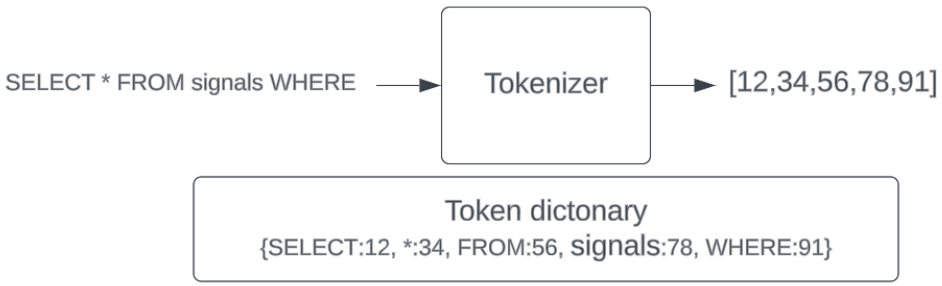

Figure 2.1: Example tokenizing the sentence "SELECT * FROM signals WHERE" into the tokens [12,34,56,78,19]. The tokens and their corresponding string are stored in a dictionary.

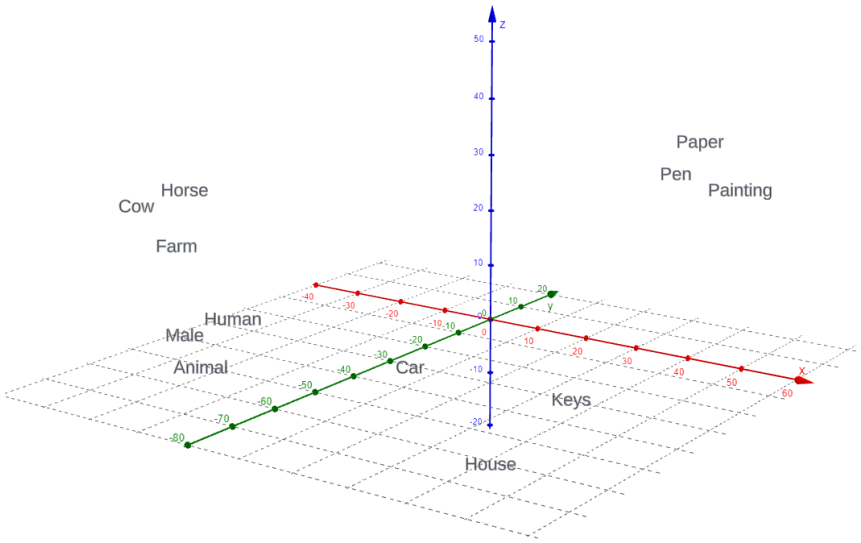

Figure 2.2: Example of possible embedding positions in 3-dimensional space.

2.2 Large Language Models

Since the paper "Attention is all you need" presented the Transformer neural network models capable of processing sequential data[9], text and language models have undergone significant advancements. Large Language models contain a profound understanding of various processing tasks, making them valuable tools for automating a wide array of problems. Among the many applications, LLMs excel in tasks such as summarizing data and generating code. The recent advent of smaller, deployable language models, like Llama 2, Deepseek, or Mistral, has facilitated finer control and adaptability in various domains, especially when considering the availability of requisite hardware for inference and fine-tuning purposes [18].

At their core, LLMs are typically composed of multiple layers of transformer blocks as depicted in 2.3. These transformer blocks consist of self-attention mechanisms and feed-forward neural networks[9]. Self-attention allows the model to weigh the importance of different words in a sentence concerning each other, efficiently capturing dependencies across the entire sequence, this is depicted in the grid 2.4. The feed-forward neural networks process the output of the attention mechanism, providing non-linear transformations to the data[3].

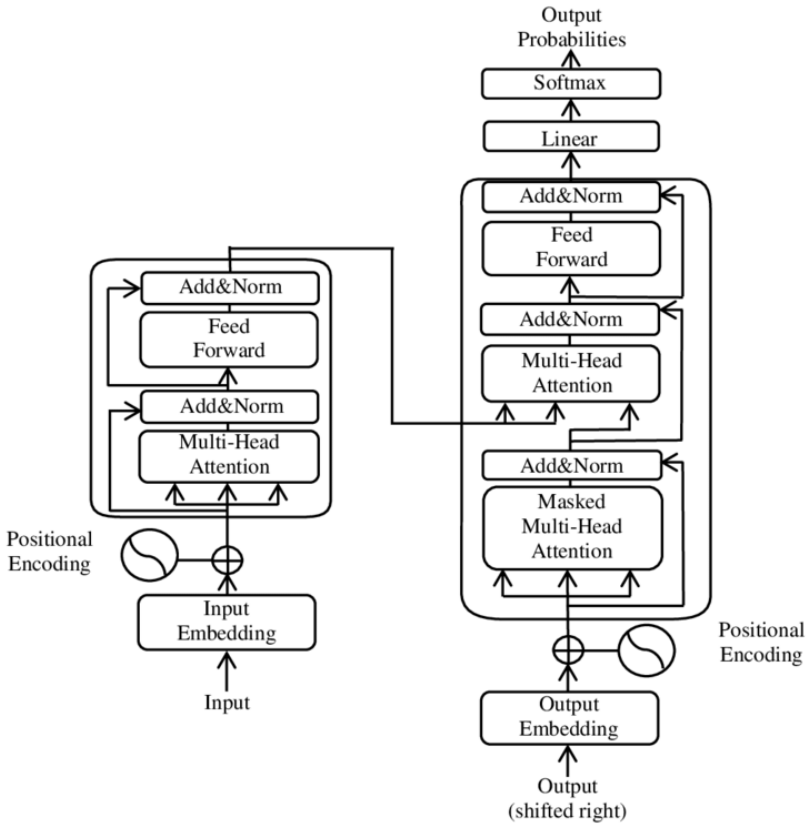

Figure 2.3: Representation of Transformer Architecture for Neural Networks. The illustration depicts the fundamental components of a transformer-based neural network model. This architecture facilitates efficient sequence processing and modeling

Language modeling is usually divided in two different tasks, one is masked language

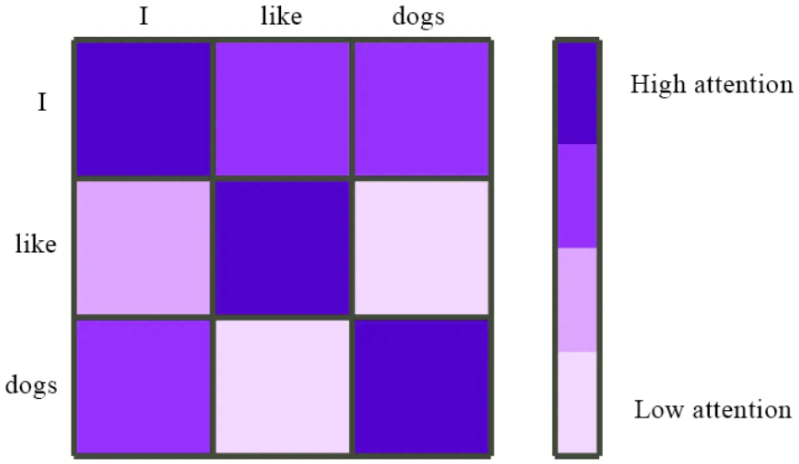

Figure 2.4: Example visualization of attention for each word in "I like dogs". The attention mechanism allows for the dynamic weighting of input features based on their relevance. It operates by assigning importance scores to different parts of the input. This grid describes how much attention the model is putting into the surrounding words when looking at each word. E.g. when the model looks at "like" it puts some attention on the word "I" and low attention on "dogs".

modeling (MLM), and the second is causal language modeling (CLM). MLM involves masking tokens from a model and asking it to predict the masked token(s). This is typically deployed for training sentence encoders such as BERT [19]. CLM is framed as a statistical word/token prediction, over some given data/training set. I.e. Given a sequence of tokens $s_1, s_2, ..., s_n$ what tokens are likely to come next in the sequence, based on the statistical patterns observed in the training data[20]. To teach this prediction to the neural network it undergoes unsupervised learning. The training process involves optimizing the model's parameters, typically using techniques such as stochastic gradient descent or its variants, to minimize the discrepancy between the predicted probabilities and the actual tokens in the training data.

Approaching the problem through this simple word prediction tends to lead to deterministic outcomes (despite it being probabilistic). In the context of code generation, this often leads the model to revert to code it has previously seen in the training data. Such behavior can be undesirable, particularly in scenarios requiring innovative solutions tailored to specific requirements. To solve this we can modify the soft-max operation in the transformer layer (2.3) with a scaling factor[9]. By increasing the scaling factor we effectively flatten the token probability curve and allow the model to be more creative[6], this is also known as the *temperature*. For programming models, a value between 0.5 and 1.5 is usually fine, with Huggingface generally recommending a value of 1.

After a model has finished training for general language tasks, it can be trained further in more specialized domains such as programming[14][6]. This can greatly increase a model's performance in these specialized tasks at the cost of versatility[5][6][7]. So called fine-tuned models can often outperform larger models, like the

model Natural-SQL (7B) outperforming the much larger GPT-3.5 (175B)[8].

2.2.1 Prompting

When asking a question to an LLM, this is called prompting the LLM. More specifically, prompting refers to the technique of providing specific cues, instructions, or examples to guide the model's generation of output. By supplying prompts, users can influence the content, style, and relevance of the model's responses, effectively directing its generation towards desired outcomes. This approach enables control and customization of LLM behavior, making it the fundamental tool in natural language processing applications.

Often the prompts and tasks are never seen before by the LLM, but they are still able to respond and sometimes even solve these tasks. This is called the zero-shot learning capabilities of the LLM. The model is able to use what it learned during training to solve new problems, and LLMs do this very well[21]. But, to further enhance performance, the adoption of few-shot learning can be applied. Few-shot prompting serves as a simpler alternative to fine-tuning, especially considering the complexities and hardware demands associated with fine-tuning [18], [22]. By providing the model with prompts containing examples of inputs and the desired outputs, one can teach the model without the need for fine-tuning[18] [22]. Brown et al.[21] showcases the efficacy of few-shot prompting. The model with few-shot achieves results comparable to models fine-tuned who received thousands of examples in their training. For more challenging tasks, the number of demonstrations can be increased, such as 3-shot, 5-shot, or 10-shot as exemplified in figure 2.5. The notable limitation of this approach is the model's token limit.

While few-shot prompting works well for many tasks, models struggle with common sense and reasoning. Wei et al.[23] introduced a solution that they call chain-of-thought prompting. This approach aims to mimic the cognitive process of human reasoning, wherein individuals build upon previous knowledge and observations to arrive at logical conclusions. The prompt instructs the model to take intermediate steps before coming to a conclusion. This makes the model able to answer questions they otherwise struggle with but humans find obvious as shown in figure 2.6.

2.3 Quantization

An issue posed with new LLMs are their size, to run and load the model each 32-bit parameter has to be loaded onto the GPU. For a 7B parameter model, this adds up to around 28GB of VRAM. Where most commercial GPUs have around 8GB of VRAM [24].

Quantization is a technique used to reduce the numerical precision of neural network parameters, thereby decreasing the computational and memory requirements of the model. This process involves representing the parameters with fewer bits than their original floating-point representations. In practice, this reduces the precision of calculations and the range of possible values, e.g. an 8-bit value can store a

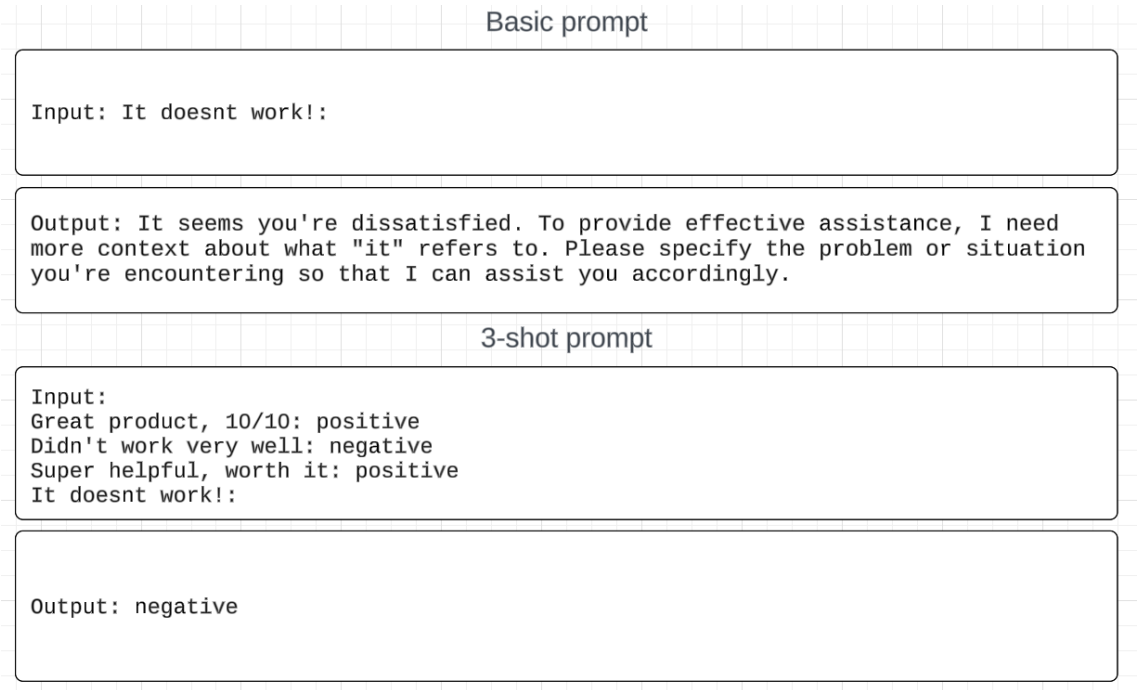

Figure 2.5: A 3-shot prompt teaches the model to perform sentiment analysis without specific instructions to do so.

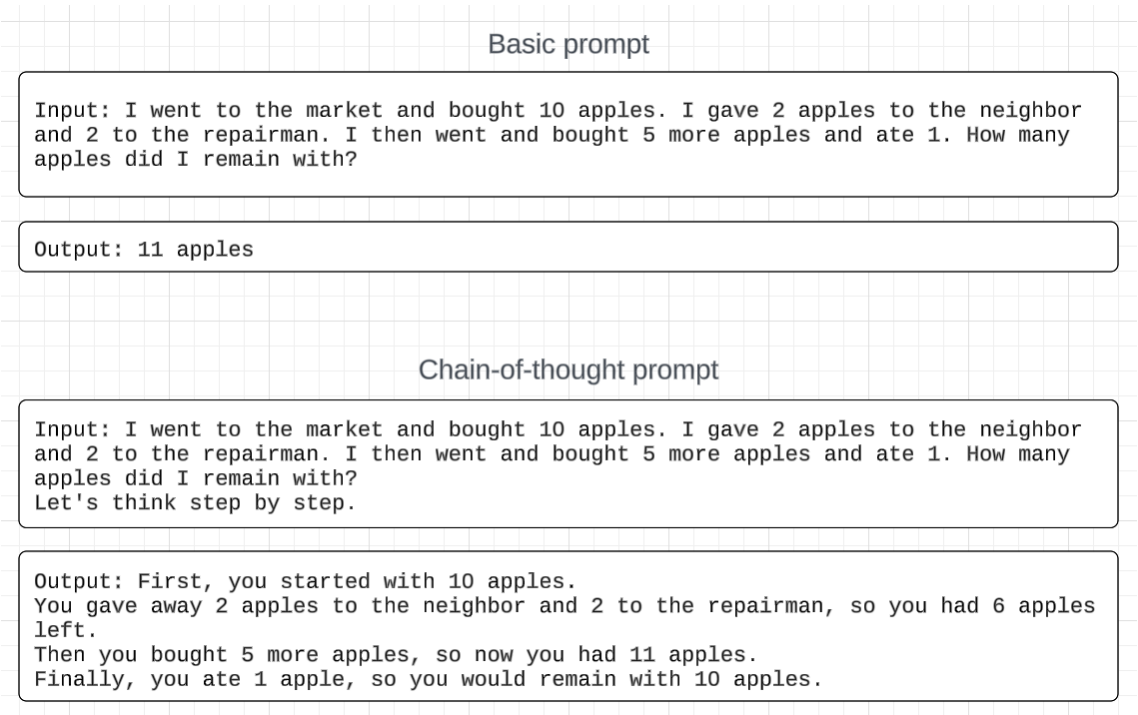

Figure 2.6: Comparison of output from a basic prompt (incorrect), with the output from a chain-of-thought prompt (correct).

maximum of 256 different values vs. a 32-bit that can store 4'294'967'296 values. By doing so, quantization aims to maintain the model's performance while reducing its computational cost and memory footprint[25]. The motivation behind quantization lies in optimizing the deployment of neural network models, this is especially useful for LLMs because of their enormous size in terms of memory. By quantizing a 32-bit 7B model to 8-bit precision, the memory requirement goes from 28GB to 7GB. Quantization introduces a trade-off between computational efficiency and model accuracy. While quantized neural network models are more resource-efficient, they may experience a degradation in performance compared to their full-precision counterparts. Therefore, quantization techniques must be carefully designed and optimized to strike the right balance between efficiency and accuracy for a given deployment scenario [26][27].

2.4 Retrieval

The token limit inherent in Large Language Models significantly constrains their ability to effectively utilize large contexts[18][3][2], a constraint that becomes more pronounced in smaller models[28]. To solve this, retrieval can be used wherein only the relevant information is retrieved and presented in the prompt. Retrieval can be solved using sentence encoders, Kiros et al. introduced an early approach for this in their work on skip-thought vectors [29]. The concept revolves around employing encoder-decoder architectures to generate embeddings for multiple entire sentences (2.7) instead of encoding single tokens, training the encoder to capture the sentences' semantic essence. The encoder-decoder framework essentially involves encoding the input sentences into fixed-length vectors, followed by decoding these vectors back into sentences. The decoded sentences is used to evaluate and perform training such as backpropagation. Building upon this foundation, Cer et al. introduced the Universal Sentence Encoder [30] utilizing the transformer architecture to achieve a, at the time, state-of-the-art result in transfer tasks. Taking this even further is BERT[19] or Bidirectional Encoder Representations from Transformers, which represents a significant advancement in natural language understanding. BERT leverages transformer with a multi-layer bidirectional encoder to capture contextual information from both the left and right contexts of each word in a sentence. The training objective of BERT involves two key tasks: masked language modeling (MLM) and next sentence prediction (NSP). In MLM, random tokens in the input sentences are masked, and BERT is trained to predict these masked tokens based on the surrounding context. This task enables BERT to learn bidirectional contextual representations of words within sentences. In NSP, BERT is trained to predict whether a sentence follows another sentence in a given text corpus. This task allows BERT to understand the relationships between sentences and grasp the overall coherence of a text. During training, BERT learns to generate contextualized embeddings for each word in a sentence, considering its context within the entire text. These embeddings capture rich semantic and syntactic information, enabling BERT to excel in a wide range of natural language understanding tasks.

The top performing encoding models on hugging face typically employ a transformer

architecture based on BERT [31]. Many of the top-performing retrieval models employ a combination of LLMs with sentence encoders to achieve a high retrieval accuracy. The disadvantage of this approach is the hardware requirements of running an entire LLM for retrieval since it is often used in tandem with another LLM. Therefore well-performing models focusing only on encoding such as BGE [32] are often preferred in a real environment, BGE-large consists of 326 million parameters which is tiny in comparison to other LLMs.

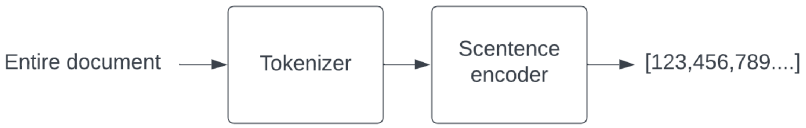

Figure 2.7: Figure showing how a document is processed and embedded into a large vector.

By encoding a database of sentences or entire documents beforehand, then at runtime encoding user queries, we can assess semantic relatedness and retrieve only the relevant context. Relatedness is measured using techniques such as Euclidean distance or cosine similarity between the sentence embeddings, facilitating information retrieval. To effectively store and search through a large database of vectors, an optimized vector storage is recommended. Facebook provides an open-source vector storage called Facebook AI Similarity Search (FAISS) [33]. It's an efficient solution for storing and finding vectors based on some similarity metric. The final retrieval framework is shown in figure 2.8

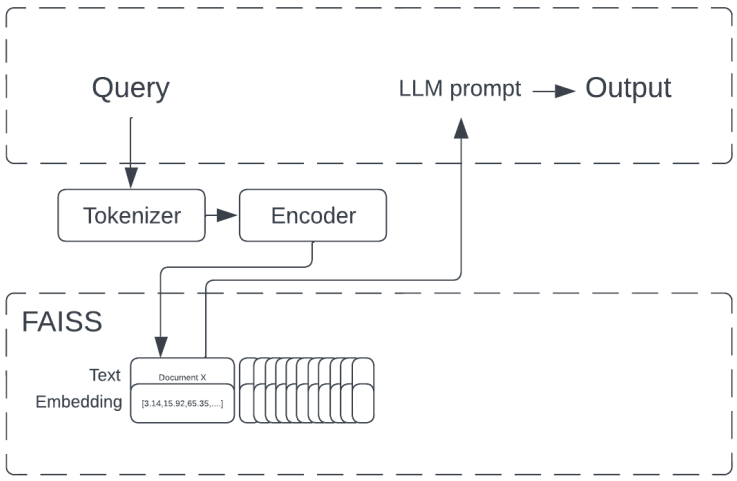

Figure 2.8: Framework of retrieval. Each context document is embedded and saved in the vector database FAISS. Then, the query is tokenized and encoded the same way as the documents in 2.7, then FAISS retrieves the n closest document encodings. The corresponding text is used as context in the LLM prompt.

2.5 Text to SQL

Developing SQL queries can be a tedious process, not only in terms of developing the code but also in the requirement for developers to acquire specific knowledge about the underlying database. Text to SQL aims to alleviate these challenges by employing a language model to interface directly with the SQL database, thereby significantly streamlining the process of data retrieval. The established systems in the benchmarks, BIRD and SPIDER, generally employ a powerful language model to generate SQL code, some intermediate steps such as fixing syntax errors, and finally prompting the database [12] [13]

To convert natural language to SQL code, Rouxi Sun et al. [34] created Fewshot SQL-PaLM and fine-tuned SQL-PaLM. To provide information for the input prompt, the data is divided into information sets, the information sets are designed to contain text information to describe the database and the user query. The database information contains data schema (tables, columns, and data types), primary and foreign keys [34]. A similar approach is presented for Dawei Gaos et als. DAIL-SQL [35]. DAIL-SQL employs similar methods to fine-tune SQL-PaLM but with more robust prompt verification checking. The performance of the systems is shown on the SPIDER dataset leaderboard. DAIL-SQL showed an 86.6% accuracy, Few-shot SQL-PaLM 77.3%, and fine-tuned SQL-PaLM 78.2% [13] [35] [34].

Fewshot SQL-PaLM, as the name suggests is presented with a long prompt containing examples of questions and answers. Each question is framed with a database schema along with a user query. For the few-shot examples correct SQL code is also provided. Then the real question is asked in the same way but without an answer[34]. This is possibly the simplest approach and there are many ways to expand upon this. DAIL-SQL iterates on this by dynamically creating the few-shot prompt by selecting the examples related to the user query from a collection of many examples. This selection process is inspired by Question Similarity Selection (QTS_S), similar to the retrieval as described in 2.4, and Masked Question Similarity Selection (MQS_S)[35] which hides the database and domain-specific tokens before encoding the examples into the database. The drawback of this approach is that it requires a large collection of examples. But in general, the current research is more focused on fine-tuning and improving LLM performance than building robust frameworks surrounding the LLM[13][7] such as error fixing or more advanced LLM chaining.

Attempts at benchmarking Text To SQL include Yu et al.[13] SPIDER Dataset. The dataset benchmarks text-to-SQL methods using Component Matching, Exact Matching, and Execution Accuracy. SQL-PaLM[34] and DAIL-SQL[35] are promising approaches for text-to-SQL. However, they assume the prompts' token sizes are within their respective model's token limit. Tables, like Zenseact's sensor data, can't fully be described within the token limit, as the column names far exceed most token limits of 4000-20000[3][6][14]. The paper from Li et al.[12] attempts to create a more realistic database, with, 95 tables and 33.4GB of dirty data. The dataset focuses on 3 main challenges: (1) handling large and dirty database values, (2) external knowledge evidence, and (3) optimizing SQL execution efficiency. Running

current text-to-SQL approaches on BIRD yields more modest results of around 50% compared to a human expert of 92% accuracy, showing that the current text-to-SQL methods are **not ready to serve as a database interface**[12].

3

Research Design

This chapter presents the research design and methodology for this thesis. The research approach selected for this project is Design Science Research (DSR). DSR is a research paradigm that combines principles from engineering and behavioral sciences to create and evaluate artifacts that solve practical problems and contribute to theory development [36].

As presented by Hevner[36], DSR follows the 7 guidelines in the table 3.1, aimed to advance the development of innovative solutions through systematic design processes, with rigorous evaluation and iteration to enhance problem-solving efficacy while addressing real-world challenges. Knauss has adapted these rules of DSR and provides advice for applying DSR on a master thesis, where academia, a final product, and evaluation are all taken into account[37]. How this project operationalizes the guidelines is described under 3.1.

Through DSR, this thesis will adhere to established theory and research to robustly create and evaluate the Columns Retrieval and Text-to-SQL systems. To ensure utility in this specific problem domain, a case study will be carried out in Zenseact's Environment.

3.1 Operationalizing Guidelines

Guideline 1: This thesis will research, develop, and refine a text-to-SQL program for querying Zenseact's tabular data.

Guideline 2: Stakeholders at Zenseact state that the data in the underutilized tables can contain useful information to improve their development. To use the data effectively the developed artifact aims to alleviate the pains of manually developing SQL queries for the sensor data table.

Guideline 3: The artifact's performance in retrieval accuracy and SQL code generation reliability will be thoroughly assessed. Specifically, the evaluation will focus on measuring the artifact's ability to retrieve relevant columns from large SQL schemas and generate errorless SQL code. The SQL code will be analyzed and compared with the SQL queries developed by Zenseact's developers to determine its understanding of the database.

Guideline 4: The developed artifact introduces a novel method for applying text-to-

Table 3.1: Hevner's DSR Guidelines [36]

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

SQL on SQL schemas too large to be described in the typical LLM's token limit. The method involves using retrieval methods to retrieve only columns relevant to the user query when describing the SQL schema.

Guideline 5: The artifact's framework is based upon established research such as SQL-PaLM [34] and tries to improve on their shortcomings described in BIRD[12]. The evaluation is done along with experienced developers at Zenseact to ensure rigorous, relevant, and correct evaluation.

Guideline 6: The artifact is developed at Zenseact on their systems alongside stakeholders and developers. Regular meetings are held similarly to Agile development [38]. This will ensure the artifact can run well on their systems as well as ensuring stakeholder satisfaction.

Guideline 7: The findings in this thesis are targeted towards developers as well as stakeholders at Zenseact, groups where the understanding of AI systems is limited. This communication will facilitate informed decision-making processes regarding the adoption and utilization of the artifact in real-world scenarios.

3.2 Problem

The overarching problem addressed in this research is twofold: (1) Zenseact wants to make use of the sensor data table for development. To effectively use the table developers have to know how to develop SQL queries, what signals exist in the 5677 column table, and what each signal value represents. When these prerequisites are taught they still have to actually develop each SQL query. This is a time consuming task taking away valuable time from developers. (2) The text-to-SQL systems written about in the current literature assume that the SQL-schema is within the bounds of the LLMs token limit which, depending on the LLM and SQL database, is not always the case. To make use of LLMs to solve problem (1), an approach to overcome the token limit must be devised.

These 2 problems are interlinked with the research questions presented in 1.2. To solve (1), RQ 2 suggests to use LLMs to generate SQL queries, and evaluate their potential. For (2), RQ 1 suggests that retrieval methods similar to RAGs be used to enable text-to-SQL on SQL databases whose schemas are larger than the LLM's token limit.

3.3 Evaluation

Zenseact has compiled a test set of 11 relevant text prompts paired with SQL code that is used by the developers. These will be used to evaluate the retrieval and SQL code generation.

For retrieval, the prompts are used to retrieve the top-k column names from the SQL schema, these columns are saved, and an expert at Zenseact grade the relevancy of these columns and also note any important left-out columns in a Case Study A.1.

The results are used to determine how well simple retrieval works for table column names, instead of the more common practice of returning documents containing plain text. For a further, less in-depth evaluation, we will compare the retrieved columns with those in the correct SQL queries provided by Zenseact. From this, the overall performance of the retrieval system can motivated.

To evaluate the performance of the same prompts but for the entire framework, first, the general reliability of the system generating running SQL code is measured in a controlled experiment. **Errorless Code Ratio** (EC) measures how often the generated code can run on the database while ignoring the output. This measures the model's capacity to generate code and use the correct table and relevant column names. Samples of the generated errorless code will be compared with the human-developed code. An informed argument will be drawn on the general quality of the code, how well the LLM understood the task and query, and whether it can select relevant columns and define conditions.

3.4 Iteration Cycles

The design science research (DSR) process is inherently iterative, allowing for continuous refinement and improvement of the artifact based on evaluation outcomes. This section outlines the planned iteration cycles for the development and enhancement of the proposed solutions, focusing on both initial design and subsequent refinement.

3.4.1 Design Cycle

The first iteration cycle will primarily focus on designing and evaluating the initial versions of the proposed artifact, encompassing both the Columns Retrieval and Text-to-SQL systems.

The effectiveness and efficiency of a basic retrieval mechanism using solely an encoder model and proximity search will be assessed. The performance of the Text-to-SQL framework, particularly its reliability in generating errorless SQL code, will be evaluated. The Errorless Code Ratio (EC) metric will be utilized to measure the proportion of generated code that can successfully execute on the database without errors. Evaluation results will be thoroughly analyzed to identify, patterns, areas requiring attention or enhancement, and what parameters to keep when proceeding with the refinement cycle.

3.4.2 Refinement Cycle

Building upon the insights gained from the initial design cycle and evaluation, the refinement cycle will focus on improving the artifact with a particular emphasis on enhancing the performance of the retrieval mechanisms to make it easier for the LLM, the retrieval will be expanded to include the retrieval paired with an LLM to further refine the selection of columns. The results from the Design Cycle will be compared with the new system, analyzing any change in the retrieval, EC, and code generation.

This research leaves room for further refinement cycles focusing on benchmarking and improving specific areas of the artifact depending on their performance or lack thereof.

4

Artifact

This chapter will detail the implementation of the artifact developed for this study. The artifact is developed to interface with the AI models to facilitate the testing of retrieval and SQL-generation. The artifacts tasks are split up between 4 big components, these components tasks remain the same over the entire development but their internal logic will be refined. The components consist of (1) Model hosting, (2) column retrieval, (3) prompt generation, and (4) SQL generation. First, this chapter will go over the selected LLM and encoding model. Each component will be described in this chapter along with the big changes during development. The model hosting machine will be referred to as *server*, and the user's machine as *client*.

It's important to note that no user interface development was developed for the artifact. This omission emphasizes that the report aims to present and evaluate methods for implementing text-to-SQL on extensive tabular data rather than developing a production software.

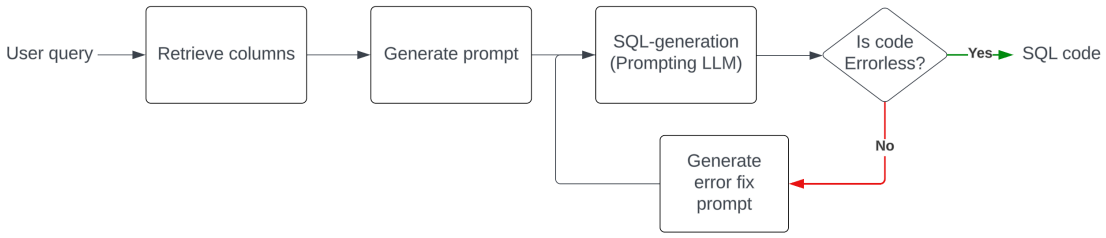

Figure 4.1: The artifact's process a given user query to SQL-code

4.1 AI Models

The models used in this artifact are the LLM Natural-SQL[7], and the encoding model BGE-large[32]. Both models were selected due to their high performance on benchmarks and leaderboards and ease of use making them a likely choice in real-world applications. Natural-SQL is a fine-tuned version of Deepseek-coder 7B that performs at the top of many coding benchmarks compared to other 7B models. Natural-SQL is also one of the few models whose only purpose is SQL generation.

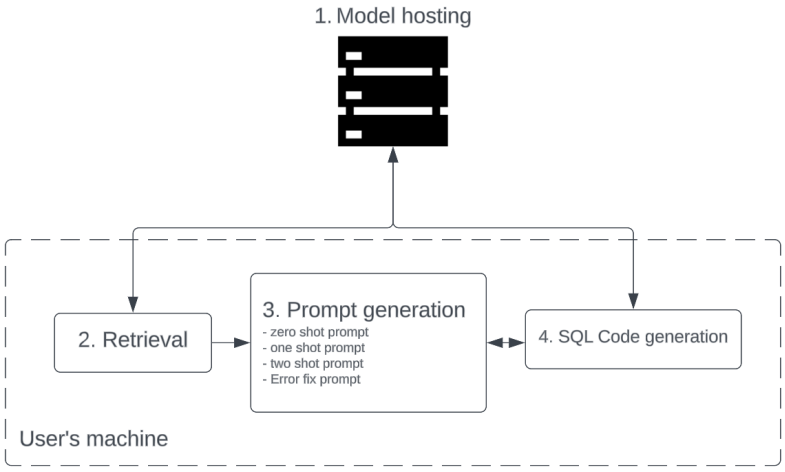

Figure 4.2: The structure of the artifact and its 4 main components: (1) Model hosting, (2) column retrieval, (3) prompt generation, and (4) SQL Code generation.

In the refinement cycle, to describe the of column names for the encoding model, Mistral-7B-Instruct[39] is deployed. At the time, this is a high-performing general-purpose LLM with possibly a better understanding of the abbreviations and technicalities of the columns' names.

4.2 Model hosting

The server, provided by Zenseact, is equipped with a 16GB Nvidia GPU. To efficiently switch between text generation and embedding tasks, a client can request a list of available models and swap loaded models at runtime. For text generation, the client sends a chat or embed request along with a prompt via the API. Subsequently, the server returns the corresponding output. While the server runs a task, such as text generation, it issues a wait code to all clients attempting to use it. This enables the client to handle scenarios where waiting is necessary. An alternative approach could involve queuing client requests, but the wait code was chosen for its simplicity.

Based on the server testing, it is determined that a 7B LLM can be operated with 8-bit quantization, loading the model takes around 1 minute. On average, generating a prompt takes 1 minute. However, if the input exceeds 340 tokens, they consume all of the GPU's memory. The 4-bit quantization allows the model to reach the token limit of Natural-SQL, which is 4096 tokens, without encountering memory issues. Generating a response using the few-shot prompts and user queries with the 4-bit model takes approximately 1 minute.

Additionally, the embedding model BGE-large operates seamlessly without any compromises. The server also allows the embedding model to be loaded on the CPU, this is to avoid having to unload and then reload the LLM every time we use the embedding since they are often used in tandem. Embedding a query takes less than

a second even for the CPU, a negligible amount of time compared to the LLM's runtime.

4.3 Retrieval

The names of the table columns are stored on the client's machine, these names are in a preprocessing step transmitted to the server, where they undergo embedding before deployment of the artifact. This embedding, coupled with the query, is then stored within a CSV file on the client's machine. The FAISS database, upon initialization, reads and loads this dictionary, enabling the retrieval functionality. To use the retrieval functionality, the client sends the new user query to the server. The server returns the embedded vector of this query, which is used to query the FAISS database and return the 20 closest signal names.

4.4 Prompt generation

To effectively generate prompts, several templates are created. The prompt is split into four parts, instructions, context, rules, and user query. The instructions describe the task and what its purpose is, and the context provides the schema. The rules explicitly tell the model important information it must follow. For few-shot prompts, the example queries and answers are appended to the prompt before the user query. Each of these parts can be exchanged depending on the task, such as the instructions being changed between SQL code generation in figure 4.3 and column selection in figure 4.5.

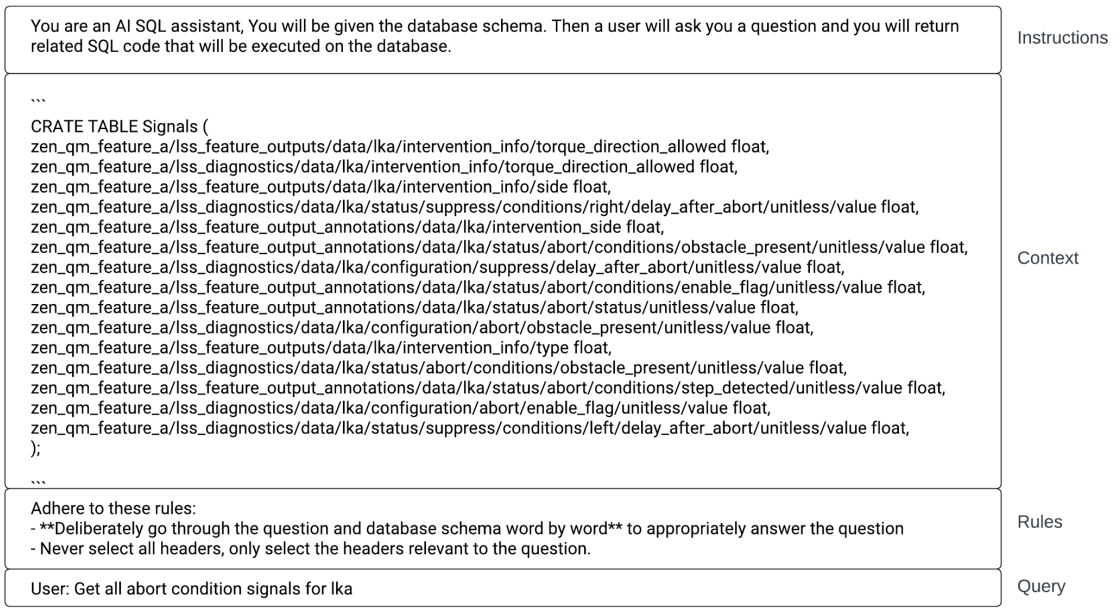

Figure 4.3: Example of zero-shot SQL generation prompt split into four parts, instructions, context, rules, and query

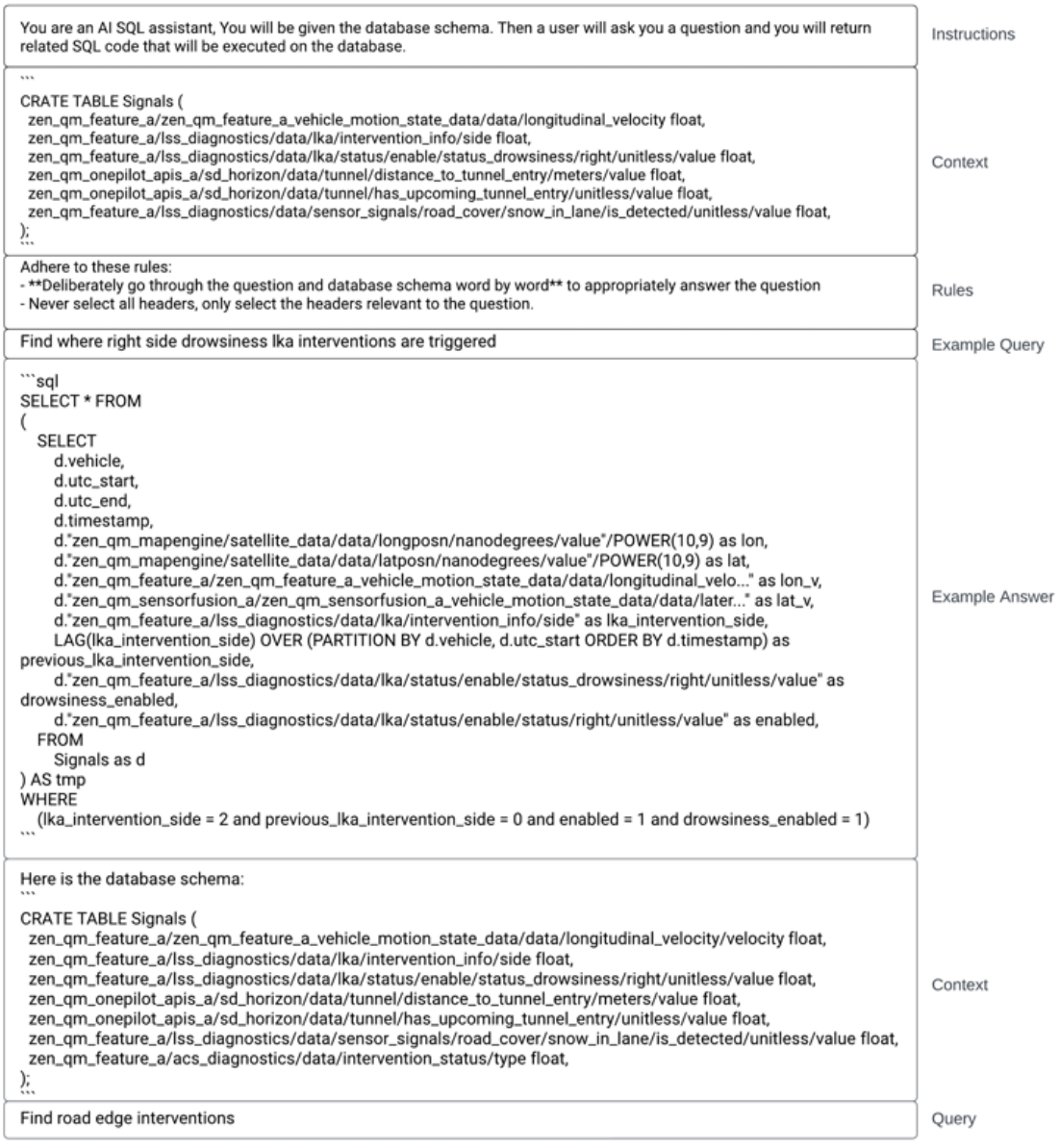

Figure 4.4: Example of one-shot SQL generation prompt split into its parts, instructions, context, rules, the example, and the user's query

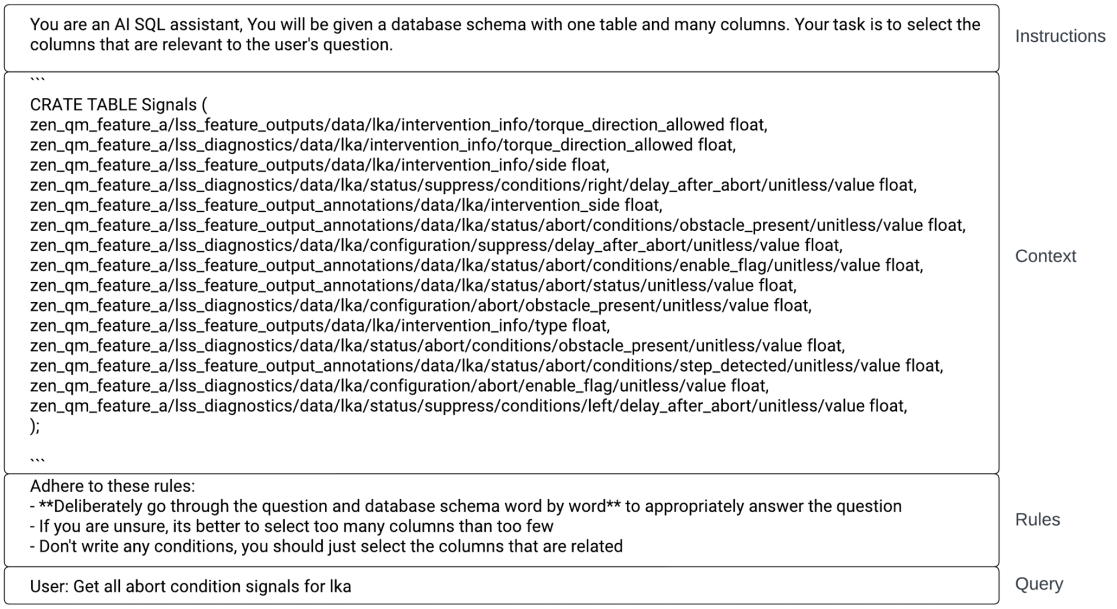

Figure 4.5: Example of select columns prompt split into four parts, instructions, context, rules, and query

4.5 SQL Code generation

The SQL code generation component takes the finalized prompt and forwards it to the LLM. Upon receiving the generated SQL code from the LLM, the system attempts to execute it on the database. If an error occurs during execution, the system forwards the error back to the prompt generation component. Subsequently, the erroneous code with an error fix prompt is sent back to the LLM for refinement. This cycle iterates until the generated SQL code successfully executes on the database or until it reaches a predefined maximum number of attempts.

From the findings in the design cycle, the SQL generation was split into two parts. The first step involves identifying and selecting the relevant columns without generating any conditions, then only these selected columns are used in the SQL generation prompt. This simplifies the LLM's job by minimizing each task's complexity, improving overall EC performance.

The iterative process ensures that the SQL code generated by the LLM meets the requirements and constraints specified in the prompt. By allowing for feedback loops between SQL generation and error handling, the system can refine and improve the quality of the generated SQL code over multiple iterations.

5

Findings

5.1 Prerequisites

Before running the benchmarks it stands to determine, what quantization level and how many columns to retrieve. This was done through basic trial and error, by attempting to run the model and seeing if it ran out of memory or not, or if the 4096 token limit was reached. The model can load at a quantization level of 8-bit, but it runs out of memory if the input tokens rise above 340, therefore a quantization level of 4-bit is used. A similar approach was made to determine the number of columns, 20 columns was the maximum amount of tokens that would fit within the model's 4096 token limit when using 2-shot prompting. With these parameters, the model runs on the in-house Nvidia Tesla T4 undisturbed.

Finally, before deploying the artifact, SQL-eval was adapted to, instead of using a local model, communicate with the server hosting our quantized model. This was done to get a comparative performance when the model is 4-bit quantized vs no quantization and to ensure everything works during longer workloads. The results from this were around 70% after running the benchmark twice. It took around 1.5 hours to complete the SQL-eval benchmark's 200 queries.

5.2 Design Cycle

The findings from the design cycle will be presented in two parts *Column retrieval* related to **RQ 1** and *SQL code generation* related to **RQ 2**. In Column retrieval, the findings from the form in A.1 are presented and the retrieval performance is analyzed. In SQL code generation the resulting Errorless Code ratio (EC) for each n-shot is presented, as well as in-depth per query EC. Finally, a table showing how often the LLM was able to fix errors in its code.

5.2.1 Column retrieval

The form in A.1 was sent out with 3 questions, "Find where left side emergency lka interventions are triggered" (lka is lane keep assist), "Find all cases where the ldw trigger goes high?" (ldw is lane departure warning), and "Find road edge interventions", accompanied by the 20 retrieved columns that the retrieval system returned.

Due to the lack of expertise and available time, one expert graded 3 questions. The results are shown in the table 5.1.

Questions	Score	Missing columns
Find where left side emergency lka interventions are triggered	3	*zen_qm_feature_a/lss_diagnostics/data/lka /status/enable/status_emergency/right/unitless /value*, *zen_qm_feature_a/lss_diagnostics/data/lka/ status/enable/status_emergency/left/unitless /value*
Find all cases where the ldw trigger goes high?	5	-
Find road edge interventions	1	*zen_qm_feature_a/acs_diagnostics/intervention _status/intervention_manager_state*

Table 5.1: Results from retrieval evaluation form A.1. Ranked on a scale of 1 bad, to 5 perfect, along with the missing signals.

For further evaluation of column retrieval, we compare the retrieved columns with that of the correct SQL-queries. There are some columns, such as car start time or car id, that are present in every SQL-queries, these are taught through the few-shot and does not have to be retrieved. To create the benchmark, each correct column is paired with its given user query, this is one test. For each user query there are 1-3 columns that are in the correct SQL-query, giving us 1-3 tests per query, or 18 tests in total.

Benchmarks:

Query:	Correct column:
Test 1: Find_where_right_side_drowsiness_lka_interventions_are_triggered	zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side
Test 2: Find_where_right_side_drowsiness_lka_interventions_are_triggered	zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_drowsiness/right/unitless/value
Test 3: Find_where_right_side_drowsiness_lka_interventions_are_triggered	zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status/right/unitless/value
Test 4: Find_all_cases_where_the_ldw_trigger_goes_high	zen_qm_feature_a/lss_diagnostics/data/ldw/warning_side_request
Test 5:	
...	
Test 18: ...	

Figure 5.1: Table of tests in the retrieval benchmark. Each query in each test is run on the retrieval system, if the "Correct column" is present in the set of retrieved columns, it is counted as correct.

5.2.2 SQL code generation

The overall EC result for all n-shots shown in table 5.2 is below 40%. This means more than half of the generated SQL code can't run on the database. The EC improves slightly as the n-shot increases reinforcing the known notion that n-shot is an effective means of prompting. Comparing the average EC per query in 5.6 shows that the hardest questions with the lowest EC are the longer queries with more than one condition (e.g. driving on a highway above 50 miles per hour while its raining).

But the longer queries' EC showed a significant increase when going from 0- to 2-shot prompting. The overall EC between 0- and 1-shot remained similar, within 2% units.

The ratios between the n-shots in the error fixing table 5.7 are very similar, around 80% of the code that ran, ran in the first iteration, and the remaining 20% ran after 1 error fix iteration. If the error was not solved in the first error fix, the LLM was not able to solve the error given more iterations.

The 3 queries with the highest overall EC were "Find all cases where the ldw trigger goes high", "Find where left side emergency lka interventions are triggered" and "Find where right side drowsiness lka interventions are triggered" as shown in 5.6. By comparing samples of generated SQL code that ran, with the code provided by Zenseact informed arguments about the quality of the generated code can be drawn, see Appendix A.2.2

Starting with the 0-shot code. The generated code is different each time, and the conditions are seemingly random. Conditions are different from the correct code and often don't contain the correct columns in them.

Find all cases where the ldw trigger goes high: In the figure A.7 in the appendix the model correctly identifies the need to look for non-zero values, but it does not use the previous iteration's value to determine when the value changes from zero to non-zero. It also wrote the condition with the wrong although similar column to the correct one. In the Appendix figures A.8 and A.9, the same column is selected, but the conditions are different and less logical.

Find where left side emergency lka interventions are triggered: In the figure A.3 the model correctly recognizes the need to check the previous value to track a trigger, but it also adds unnecessary logic regarding road crossings. In figures A.1 and A.2 the model generated complex and incorrect conditions.

Find where right side drowsiness lka interventions are triggered: The model only uses the drowsiness column in the condition in figure A.5, but it also adds incorrect and unnecessary logic. In figures A.4 and A.6 the model produced incorrect code.

The following paragraphs discuss the 1-shot code. The code is categorized as being very short. It prefers generating short SQL queries that return all columns.

Find all cases where the ldw trigger goes high: In figures A.16, A.17, and A.18, the code presented is very short compared to other outputs. It also chooses to select all 5677 columns in the table which is typically not what you want, both in terms of database performance and readability of the output. The conditions in the code are also incorrect.

Find where left side emergency lka interventions are triggered: The code in figures A.10, A.11, and A.12 is, similarly to the results in *Find all cases where the ldw trigger goes high* above, brief and selects all columns from the table.

Find where right side drowsiness lka interventions are triggered: Figures A.13, A.14, and A.15 showcase similar patterns to the previous examples in the 1-shot category.

Finally, the following presents the 2-shot code. It demonstrates an improvement in the selection of columns and the definition of conditions. However, the model occasionally persists in generating very short queries and selecting all columns as observed in the 1-shot code.

find all cases where the ldw trigger goes high: In the figures A.26 and A.27 the model has generated conditions checking for a value greater than zero, similar to the 0-shot version. But like it did with the 1-shot prompt, it selects all columns. In figure A.25, there's a slight improvement, but the trigger is checked for "= 1" instead of a non-zero or greater than zero value. Moreover, incorrect columns are used for the condition.

find where left side emergency lka interventions are triggered: A.19 introduces a new approach in column selection, successfully figuring out how to select the previous trigger value. However, when it comes to checking the conditions, it looks for a "NULL" value instead of 0. In A.20, the model's code selects the correct columns and checks the trigger accurately. However, it fails to check if the diagnostics are enabled and if the signal for emergency is triggered. Meanwhile, in A.21 the model successfully checks the emergency but neglects the rest.

find where right side drowsiness lka interventions are triggered: the generated code varies greatly in length. The code and conditions in A.23 are close to being correct, but it contains an unnecessary "or" statement checking for when the drowsiness status is off. Apart from that, it correctly selects the columns and checks the triggers and previous values. The generated code in A.22 and A.24 does not check for drowsiness and opts to select all columns.

To draw some conclusions about what happens when the code fails to run, we analyze samples of the erroneous code. The generated SQL queries where sampled from the same 3 queries, for 0,1, and 2 shot, but this time the SQL queries where not able to run on the database. Out of the 9 samples A.28-36, 6 of the errors where caused by a "not found" or "does not exist", caused by either a typo or a hallucinated column or table names. In A.33 the model generated a description of the solution in text which caused a syntax error. In A.36 there was a simple syntax error where the model forgot to encapsulate the column in quotation marks.

n-shot	EC
0	$\frac{126}{350} = 0.36$
1	$\frac{122}{350} = 0.35$
2	$\frac{138}{350} = 0.39$

Table 5.2: Table showing Errorless Code ratio (EC) for 0, 1, and 2-shot

5.2.3 Conclusion design cycle

Overall the findings are underwhelming. However, the findings showed a marginal improvement from 0 to 2-shot, both in logic and EC. Error fixing is worth keeping but can be lowered to just 1 attempt at fixing the error. The main issues stem from,

Query	EC
Find where left side emergency lka interventions are triggered	59.26%
Find where right side drowsiness lka interventions are triggered	52.63%
Find road edge interventions	48.00%
Get all enable condition signals for right side ldw	22.22%
Get all abort condition signals for lka	42.31%
Get all suppression condition signals for left side ldw	4.76%
Get all trigger condition signals for left side lka	39.29%
Find all cases where the ldw trigger goes high	65.22%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit changed from 25 meters per second to 15 meters per second	6.45%
Events where driver overrides feature acceleration	36.36%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit spiked to some value that was held for less than four seconds	3.23%

Table 5.3: The EC per query for all 0-shots.

Query	EC
Find where left side emergency lka interventions are triggered	46.15%
Find where right side drowsiness lka interventions are triggered	40.32%
Find road edge interventions	35.29%
Get all enable condition signals for right side ldw	35.71%
Get all abort condition signals for lka	37.84%
Get all suppression condition signals for left side ldw	24.24%
Get all trigger condition signals for left side lka	31.71%
Find all cases where the ldw trigger goes high	59.26%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit changed from 25 meters per second to 15 meters per second	14.81%
Events where driver overrides feature acceleration	48.48%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit spiked to some value that was held for less than four seconds	0.00%

Table 5.4: The EC per query for all 1-shots. The **highlighted** query was used in the few-shot prompt.

Query	EC
Find where left side emergency lka interventions are triggered	56.52%
Find where right side drowsiness lka interventions are triggered	48.00%
Find road edge interventions	39.29%
Get all enable condition signals for right side ldw	29.41%
Get all abort condition signals for lka	50.00%
Get all suppression condition signals for left side ldw	35.29%
Get all trigger condition signals for left side lka	42.11%
Find all cases where the ldw trigger goes high	61.11%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit changed from 25 meters per second to 15 meters per second	15.15%
Events where driver overrides feature acceleration	42.86%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit spiked to some value that was held for less than four seconds	10.00%

Table 5.5: The EC per query for all 2-shots. The **highlighted** queries were used in the few-shot prompt.

Query	EC
Find where left side emergency lka interventions are triggered	53.41%
Find where right side drowsiness lka interventions are triggered	46.68%
Find road edge interventions	41.13%
Get all enable condition signals for right side ldw	28.32%
Get all abort condition signals for lka	43.38%
Get all suppression condition signals for left side ldw	21.92%
Get all trigger condition signals for left side lka	37.70%
Find all cases where the ldw trigger goes high	61.65%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit changed from 25 meters per second to 15 meters per second	11.65%
Events where driver overrides feature acceleration	43.07%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit spiked to some value that was held for less than four seconds	4.41%

Table 5.6: The average overall EC for all n-shot.

n-shot	0 fixes	1 fix	2 fixes	sum
0	90	21	0	111
1	103	27	0	130
2	113	28	0	141

Table 5.7: Table showing the number of times code executed with no errors after 0, 1, and 2 fixes

```
SELECT * FROM
(
    SELECT
        d.vehicle,
        d.utc_start,
        d.utc_end,
        d.timestamp,
        d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
        d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
        d."zen_qm_feature_a/zen_qm_feature_a_vehicle_motion_state_data/data/longitudinal_velocity/velocity/
            meters_per_second/value" as lon_v,
        d."zen_qm_sensorfusion_a/zen_qm_sensorfusion_a_vehicle_motion_state_data/data/lateral_velocity/
            velocity/meters_per_second/value" as lat_v,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side" as lka_intervention_side,
        LAG(lka_intervention_side) OVER (PARTITION BY d.vehicle, d.utc_start ORDER BY d.timestamp) as
            previous_lka_intervention_side,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_emergency/left/unitless/value" as
             emergency_enabled,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status/left/unitless/value" as enabled,
    FROM
        read_parquet('parquet_files/{firezone_to_query}/{resim_version}/data/*.parquet.gzip') as d
) AS tmp
WHERE
    (lka_intervention_side = 1 and previous_lka_intervention_side = 0 and enabled = 1 and emergency_enabled
        = 1)
```

Figure 5.2: Correct SQL code for "Find where left side emergency lka interventions are triggered"

```
SELECT * FROM
(
    SELECT
        d.vehicle,
        d.utc_start,
        d.utc_end,
        d.timestamp,
        d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
        d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
        d."zen_qm_feature_a/zen_qm_feature_a_vehicle_motion_state_data/data/longitudinal_velocity/velocity/
            meters_per_second/value" as lon_v,
        d."zen_qm_sensorfusion_a/zen_qm_sensorfusion_a_vehicle_motion_state_data/data/lateral_velocity/
            velocity/meters_per_second/value" as lat_v,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side" as lka_intervention_side,
        LAG(lka_intervention_side) OVER (PARTITION BY d.vehicle, d.utc_start ORDER BY d.timestamp) as
            previous_lka_intervention_side,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_drowsiness/right/unitless/value"
            as drowsiness_enabled,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status/right/unitless/value" as enabled,
    FROM
        read_parquet('parquet_files/{firezone_to_query}/{resim_version}/data/*.parquet.gzip') as d
) AS tmp
WHERE
    (lka_intervention_side = 2 and previous_lka_intervention_side = 0 and enabled = 1 and
         drowsiness_enabled = 1)
```

Figure 5.3: Correct SQL code for "Find where right side drowsiness lka interventions are triggered"

```
SELECT * FROM
(
    SELECT
        d.vehicle,
        d.utc_start,
        d.utc_end,
        d.timestamp,
        d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
        d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
        d."zen_qm_feature_a/zen_qm_feature_a_vehicle_motion_state_data/data/longitudinal_velocity/velocity/
            meters_per_second/value" as lon_v,
        d."zen_qm_sensorfusion_a/zen_qm_sensorfusion_a_vehicle_motion_state_data/data/lateral_velocity/
            velocity/meters_per_second/value" as lat_v,
        d."zen_qm_feature_a/lss_diagnostics/data/ldw/warning_side_request" as ldw_left_trigg,
        LAG(ldw_left_trigg) OVER (PARTITION BY d.vehicle, d.utc_start ORDER BY d.timestamp) as
            previous_ldw_left_trigg,
    FROM
        read_parquet('parquet_files/{firezone_to_query}/{resim_version}/data/*.parquet.gzip') as d
) AS tmp
WHERE
    (ldw_left_trigg != 0 and previous_ldw_left_trigg = 0)
```

Figure 5.4: Correct SQL code for "Find all cases where the ldw trigger goes high"

selecting the correct columns, defining conditions, using the correct columns for the conditions, and from analyzing the errors copying the columns without any mistakes is difficult. These are the issues that will be addressed in the refinement cycle.

The column retrieval performed similarly, from the limited evaluation by the developers at Zenseact, the retrieval was usable 2 out of 3 times. From our own comparison with the correct columns, we got 4 out of the 18 columns. We don't know if all of these 18 columns are absolutely necessary to develop a sufficient query, but there is room for improvement. What's likely the case is that many of the long column names contain similar words e.g. 300 of the columns beginning with "zen_qm_feature_a/lss_diagnostics/data/lka/...", making it difficult for the encoding model to differentiate between the columns and the small differences between them. This could also be another reason why 2-shot performed better, the LLM can use the columns provided in the few-shot examples even if they are not present in the retrieved columns.

5.3 Refinement Cycle

The column retrieval performance was mediocre, and the overall artifact performed badly. The problems the LLM was presented with were too complicated and the long column names further confuse the LLM. A way to simplify the problem is to give the LLM fewer columns to work with, but then we have to be certain the important columns won't be missing when asking the LLM to generate the code. We attempt to do this by splitting the job of generating the SQL queries into two steps, the first is selecting columns to use, and the second is to write the code using the selected columns.

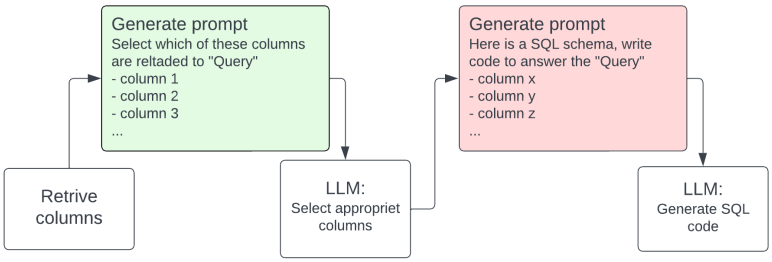

Figure 5.5: Figure describes how the problem is divided into multiple steps. The columns are retrieved as normal, but before the code is generated, the selection of columns is refined using another LLM.

To address the issue of selecting all, the SQL-generation prompt will be changed to include "Never select all columns, only select the columns relevant to the question".

Then to refine the retrieval, before embedding the columns, we will let the LLM Mistral 7B, which is a non-specialised LLM, explain each column. Then get the embedding vectors using these explanations instead of the column. This will hopefully get around the issues of column names using similar words. In simpler terms, we

will search for columns using each columns explanation instead of their names. This will result in two tests, one where only the comments are embedded and one where both the column name and headers are embedded together.

The findings from the refinement cycle are shown under *Retrieval* and *SQL code generation*. The EC will be presented per query for the two-shot prompt. Finally, the section presents the errorless generated code from the same 3 queries in the design cycle and determines its correctness.

5.3.1 Retrieval

The first case, where we only embed the generated comment, the result was 2 out of 18, which is worse than the design cycle with just the column names. When embedding the column name along with the comment, the score was 4 out of 18. The SQL-code generation will be evaluated using both the embedded column name with comment and only column name to see if there is any performance difference between the two.

Figure 5.6: A column along with a LLM generated explanation.

5.3.2 SQL code generation

With the simplified columns selection the EC increased from 39% to over 57% for 2-shot prompting. The EC per query shown in table 5.11 displays a similar story to the design cycle where the longer questions are harder. The utility of the error fix in table 5.12 decreased slightly from the design cycle where 20% of the code that ran was fixed, after the refinements this number is now 15%. The result for commented and not commented columns when using retrieval was the same when benchmarking SQL code generation.

The SQL queries are still different in every sample and the new prompt didn't manage to eliminate the model from using SELECT * (select all) even though the prompt instructed it not to. Similarly, issues repeat themselves in the conditions as well, it doesn't properly check triggers and tends to write long, complicated, and illogical conditions. The overall code quality is poor.

n-shot	EC
2	$\frac{198}{350} = 0.57$

Table 5.8: Table showing Errorless Code ratio (EC) for 2-shot after refinement using commented columns in retrieval.

5.3.3 Conclusion refinement cycle

While the EC improved, the code quality did not, nor did it become remarkably worse. On the SQL-eval benchmark, our LLM setup, without retrieval, achieves

n-shot	EC
2	$\frac{202}{350} = 0.58$

Table 5.9: Table showing Errorless Code ratio (EC) for the first 350 test for 2-shot after refinement using non-commented columns in retrieval.

n-shot	EC
2	$\frac{485}{847} = 0.57$

Table 5.10: Table showing Errorless Code ratio (EC) for 2-shot after refinement using non-commented columns in retrieval.

Query	EC
Find where left side emergency lka interventions are triggered	73.13%
Find where right side drowsiness lka interventions are triggered	70.63%
Find road edge interventions	67.61%
Get all enable condition signals for right side ldw	65.63%
Get all abort condition signals for lka	79.71%
Get all suppression condition signals for left side ldw	84.06%
Get all trigger condition signals for left side lka	32.88%
Find all cases where the ldw trigger goes high	70.83%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit changed from 25 meters per second to 15 meters per second	17.19%
Events where driver overrides feature acceleration	50.00%
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit spiked to some value that was held for less than four seconds	10.13%

Table 5.11: The EC per query for all 2-shots. The **highlighted** queries were used in the few-shot prompt.

n-shot	0 fixes	1 fix	sum
2	415	70	485

Table 5.12: Table showing the number of times code executed with no errors after 0 and 1 fixes.

72% accuracy when checking if the SQL query retrieves the correct columns. The quantized model achieves a higher percentage of correct code on the SQL-eval benchmark than we get EC. By comparing the database in the SQL-eval benchmark with Zenseacts we can draw some conclusions on what the model struggles with and what it prefers. The column names are the main differences between SQL-eval's test and ours, examples of columns in SQL-eval [river.length, river.country_name, business.name, review.rating]. These names' meanings are obvious to any reader. It is also likely that the model has encountered these words in training and has a good idea of what they mean. This is not the case for Zenseact's sensor table, long column names with many abbreviations and required company-specific knowledge. This, of course, makes it extremely hard for the model to guess what the SQL query should be. This is also likely the cause for the seemingly random conditions in the generated SQL queries. The refined retrieval result was disappointing. But the culprit is likely a similar reason; the mentioned lack of knowledge within Zenseact's domain.

To solve this, the model somehow needs to understand the columns. This can be approached in different ways: change the names to be shorter and more descriptive, or train the model on the in-house knowledge. There are drawbacks to both of these approaches; simplifying the names is a massive undertaking that would take a lot of manpower and would change how everyone at Zenseact uses the table. To train the model, a big dataset is required to effectively capture the nuances of the column names and their meanings. However, gathering such a dataset entails significant effort and resources.

5.4 Refinement 2, Retrieval cycle

This short cycle focuses on exploring refinements to the retrieval functionality and will be benchmarked using the compare retrieved headers test. To lessen the difficulty originating from the long column names, they can be processed to remove information not necessary for retrieval. The method presented in this cycle takes inspiration from finding stop words by term frequency in NLP[40]. The column names are split by "/" into individual terms, how many times the terms appear in the columns is counted 5.7. Each term, in descended order of frequency, is removed from the columns one by one, for each removal the columns are checked if they're still distinct, if they are not distinct that term removal fails. Some words that appear high up on this frequency list, are still important to the examples queries such as "lka", "left", and "right". These terms are marked as protected and won't be removed. This list can be further optimized with the help of an experienced developer. Finally, the compression can be tuned with the parameter min_freq where only terms with a frequency higher than min_freq are attempted to be removed 5.8. With $min\_freq = 1$ the algorithm tries to remove all words, but prioritizes the most common words first, in combination with checking if the columns are distinct we approximate which words are important. The retrieval benchmark will be run for $min\_freq = [1, 10, 99, 500]$.

('data', 5674)
('value', 4838)
('zen_qm_feature_a', 3813)
('unitless', 3278)
('zen_qm_sensorfusion_a', 1217)
('status', 868)
('conditions', 710)
('lss_diagnostics', 628)
('left', 579)
('right', 579)
('enable', 559)
('lka', 540)
('meters', 512)
('zen_qm_sensorfusion_a_positioning_hdmap_debug_data', 500)
('zen_qm_onepilot_apis_a', 430)
('lss_feature_output_annotations', 391)
('nominal_motion_planner_debug', 375)
('ldw', 372)
('suppress', 266)
('measurement_model_qualities', 240)
('configuration', 235)
('acs_diagnostics', 223)
('health_monitor_a_debug', 216)
('ca_long_qm_debug', 188)

Figure 5.7: The top 24 terms and their frequencies in sensor table columns.

5.4.1 Findings

The base retrieval system with no column compression gets 4 out of 18 on the retrieval benchmark. For $min\_freq = [1, 10, 99]$ the result where 2 out of 18. For $min\_freq = 500$ the results improved to 6 out of 18.

5.4.2 Conclusion Retrieval cycle

The degree of compression is important to get right, too much compression quickly degrades performance, but compressing the columns a little bit seemingly catches and removes common terms that are not necessary to determine the columns meaning. The optimal list of protected words and the min_freq value will likely vary a lot depending on the database. There is also the potential for using more sophisticated methods to simplify the columns without removing distinctions or their meanings.

While the column compression did slightly improve performance on the small benchmark. For a more conclusive result, it would need a larger set of tests. However, the findings show merit in removing high-frequency terms to simplify columns for AI retrieval models.

```
initialize word_counter as an empty dictionary

for each column in columns:
    words := split the column into words
    for each word in words:
        if word is already in word_counter:
            increment the count of word in word_counter
        else:
            add word to word_counter with a count of 1

sorted_words = word_counter.sort(DESC)


initialize columns_clean as a copy of columns

function try_remove_word(columns: list of strings, to_remove: string) -> list of strings:
    create a copy of columns and store it in columns_temp
    for each column in columns_temp:
        if to_remove is present in column:
            column.remove(to_remove) #remove "to_remove" from "column" in "columns_temp"
            if all_distinct(columns_temp):
                columns := columns_temp
            else:
                columns_temp := columns
    return columns


min_freq = 500
protected_words = ["lka", "ldw", "left",...]

for i in length of sorted_words:
    (word,count) := sorted_words[i]

    if word is in protected_words:
        continue to the next iteration

    if count of word <= min_freq:
        exit the loop

    columns_clean := try_remove_word(columns, word)
```

Figure 5.8: Pseudo code algorithm for removing high-frequency terms/words to simplify the column names.

6

Conclusion

This thesis aimed to enhance the efficiency of querying large tabular datasets by leveraging Large Language Models (LLMs) for text-to-SQL conversion, particularly focusing on Zenseact's extensive sensor data. This chapter concludes the study by summarizing the key findings, incorporating insights from the development cycles, answering the research questions, and discussing potential areas for future research.

6.1 Summary of Findings

This research demonstrated that while LLMs have the potential to transform natural language prompts into SQL queries effectively, there are significant challenges when dealing with large and complex databases like Zenseact's. The two research questions addressed in this thesis were:

- **RQ 1:** To which degree can retrieval methods select appropriate columns in a large SQL schema to allow for text-to-SQL to be performed?
- **RQ 2:** To what extent can current open and deployable 7B LLMs work for text-to-SQL tasks?

6.1.1 Research Question 1

To answer RQ1, the study investigated the effectiveness of retrieval methods in selecting the appropriate columns for text-to-SQL tasks. The findings revealed that retrieval methods can generate an SQL schema that fits within a given model's token limit and can sometimes even improve text-to-SQL performance by narrowing the selection to only relevant columns. However, the developed artifact was lacking in some areas.

Cycles Insights:

The unmodified base performance of available embedding models, when used for retrieving relevant columns, is lacking for our task. The performance will differ depending on the embedding model and the database naming conventions. But overall, the performance on databases with complex column names requiring domain-specific knowledge won't be good.

In the retrieval's refinement cycle under section 5.4, column names were processed

to remove high-frequency terms that were deemed non-essential. This compression aimed to reduce the complexity of the column names while preserving their distinctiveness. The base retrieval system in the design cycle without column compression scored 4 out of 18 on the retrieval benchmark. When compression was applied with a minimum frequency threshold of 500, performance improved to 6 out of 18. This suggests that moderate compression can help the model understand what's important and differentiating in each column.

In conclusion, while retrieval methods show promise, their current implementation without significant domain-specific customization and fine-tuning does not perform adequately for complex SQL schemas. Further refinement and optimization are necessary to achieve satisfactory results.

6.2 Research Question 2

To address RQ2, the study evaluated the performance of the model Natural-SQL on text-to-SQL tasks. The results indicated that for simple databases Natural-SQL performed well on text-to-SQL tasks and benchmarks such as SQL-eval. However, when deployed on complex databases the model struggled, particularly on those databases with lengthy and complicated column names, similar to the ones in Zenseacts sensor data. Improving performance on these databases would require either fine-tuning an LLM on specific datasets or preprocessing the database schema to make it more manageable for the models.

Cycle Insights:

From the benchmarks SPIDER[13], SQL-eval[8], and BIRD[12] we get a good idea of the performance of the current LLMs on text-to-SQL. The Natural SQL model performs well on simple benchmarks like SQL-eval, but it would likely struggle on the BIRD benchmark like GPT-4 does. This is corroborated by the results of the initial design cycle in this report, where the longer columns cause the model to make mistakes.

From the column selection refinement in the refinement cycle, it was found that while the Errorless code ratio (EC) improved, the overall quality of the errorless code remained the same. On the SQL-eval benchmark, the LLM setup without retrieval achieved 72% accuracy which the model did not even come measurably close to on the Zenseacts database, if measured the accuracy would be close to 0%. This suggests that the model struggles with Zenseacts unique column names and abbreviations, which are probably not encountered in typical training data for an LLM.

In summary, current open-source models can handle text-to-SQL tasks on simple databases but require further refinement and adaptation to generate SQL queries for more complex databases effectively.

6.3 Recommendations for Future Research

Based on the findings, for improving a LLM's performance, future research can explore:

Developing Standardized Training Processes: Establishing a well-defined process for creating labeled datasets for training and fine-tuning encoding and Large language models specifically for tasks on large databases. This will create an easy guide for people looking to implement text-to-SQL in their services.

Custom Token Introduction: Investigating the introduction of custom tokens for columns to improve model understanding and reduce errors in complex databases. By introducing tokens derived from columns' names the model can get a better understanding of each column and its purpose. This will also diminish the number of hallucinated columns, as each column is defined as fewer tokens. This is given there is sufficient training data and labels for each column in the database.

In summary, while this study has shown the potential of LLMs in facilitating text-to-SQL conversion, significant challenges remain. Addressing these through targeted research and development will be crucial for realizing the full potential of these models in industrial applications.

Bibliography

[1] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2013. DOI: 10.1109/TSE.2012.67.

[2] OpenAI, : J. Achiam, *et al.*, *Gpt-4 technical report*, 2023. arXiv: 2303.08774 [cs.CL].

[3] J. Ye, X. Chen, N. Xu, *et al.*, *A comprehensive capability analysis of gpt-3 and gpt-3.5 series models*, 2023. arXiv: 2303.10420 [cs.CL].

[4] *Can Ai Code Results - a Hugging Face Space by mike-ravkine — huggingface.co*, https://huggingface.co/spaces/mike-ravkine/can-ai-code-results, [Accessed 16-01-2024].

[5] [Online]. Available: https://evalplus.github.io/leaderboard.html.

[6] DeepSeek, *Deepseek coder: Let the code write itself*, https://github.com/deepseek-ai/DeepSeek-Coder, 2023.

[7] Feb. 2024. [Online]. Available: https://huggingface.co/chatdb/natural-sql-7b.

[8] W. J. Ping, *Open-sourcing sqleval: Our framework for evaluating llm-generated sql*, Aug. 2023. [Online]. Available: https://defog.ai/blog/open-sourcing-sqleval/.

[9] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[10] P. Zhang, S. Xiao, Z. Liu, Z. Dou, and J.-Y. Nie, *Retrieve anything to augment large language models*, 2023. arXiv: 2310.07554 [cs.IR].

[11] X.-Y. Fu, M. T. R. Laskar, E. Khasanova, C. Chen, and S. B. TN, *Tiny titans: Can smaller large language models punch above their weight in the real world for meeting summarization?* 2024. arXiv: 2402.00841 [cs.CL].

[12] J. Li, B. Hui, G. Qu, *et al.*, *Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls*, 2023. arXiv: 2305.03111 [cs.CL].

[13] T. Yu, R. Zhang, K. Yang, *et al.*, *Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task*, 2019. arXiv: 1809.08887 [cs.CL].

[14] B. Rozière, J. Gehring, F. Gloeckle, *et al.*, *Code llama: Open foundation models for code*, 2024. arXiv: 2308.12950 [cs.CL].

[15] OpenAI, *tiktoken*, https://github.com/openai/tiktoken, Accessed: March 26, 2024, n.d.

[16] M. Staron, *Machine Learning Infrastructure and Best Practices for Software Engineers: Take your machine learning software from a prototype to a fully fledged software system*, English. Packt, 2024, ISBN: 9781837634064.

[17] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, 2013. arXiv: 1301.3781 [cs.CL].

[18] H. Touvron, L. Martin, K. Stone, *et al.*, *Llama 2: Open foundation and fine-tuned chat models*, 2023. arXiv: 2307.09288 [cs.CL].

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[20] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2018. [Online]. Available: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

[21] T. B. Brown, B. Mann, N. Ryder, *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[22] J. Kaplan, S. McCandlish, T. Henighan, *et al.*, *Scaling laws for neural language models*, 2020. arXiv: 2001.08361 [cs.LG].

[23] J. Wei, X. Wang, D. Schuurmans, *et al.*, *Chain-of-thought prompting elicits reasoning in large language models*, 2023. arXiv: 2201.11903 [cs.CL].

[24] Steam Hardware & Software Survey. "Steam Hardware Survey - Video Card." Accessed: March 26, 2024, Valve Corporation. (2024), [Online]. Available: https://store.steampowered.com/hwsurvey/videocard/.

[25] Hugging Face, *Quantization*, https://huggingface.co/docs/transformers/main_classes/quantization, Accessed on: 2024-03-24, 2024.

[26] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, *Gptq: Accurate post-training quantization for generative pre-trained transformers*, 2023. arXiv: 2210.17323 [cs.LG].

[27] Z. Liu, B. Oguz, C. Zhao, *et al.*, *Llm-qat: Data-free quantization aware training for large language models*, 2023. arXiv: 2305.17888 [cs.CL].

[28] V. Karpukhin, B. Ouz, S. Min, *et al.*, *Dense passage retrieval for open-domain question answering*, 2020. arXiv: 2004.04906 [cs.CL].

[29] R. Kiros, Y. Zhu, R. Salakhutdinov, *et al.*, *Skip-thought vectors*, 2015. arXiv: 1506.06726 [cs.CL].

[30] D. Cer, Y. Yang, S.-y. Kong, *et al.*, *Universal sentence encoder*, 2018. arXiv: 1803.11175 [cs.CL].

[31] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "Mteb: Massive text embedding benchmark," *arXiv preprint arXiv:2210.07316*, 2022. DOI: 10.48550/ARXIV.2210.07316. [Online]. Available: https://arxiv.org/abs/2210.07316.

[32] S. Xiao, Z. Liu, P. Zhang, and N. Muennighoff, *C-pack: Packaged resources to advance general chinese embedding*, 2023. arXiv: 2309.07597 [cs.CL].

[33] M. Douze, A. Guzhva, C. Deng, *et al.*, "The faiss library," 2024. arXiv: 2401.08281 [cs.LG].

[34] R. Sun, S. O. Arik, H. Nakhost, *et al.*, *Sql-palm: Improved large language model adaptation for text-to-sql*, 2023. arXiv: 2306.00739 [cs.CL].

[35] D. Gao, H. Wang, Y. Li, *et al.*, *Text-to-sql empowered by large language models: A benchmark evaluation*, 2023. arXiv: 2308.15363 [cs.DB].

[36] A. Hevner, A. R, S. March, *et al.*, "Design science in information systems research," *Management Information Systems Quarterly*, vol. 28, pp. 75–, Mar. 2004.

[37] E. Knauss, *Constructive master's thesis work in industry: Guidelines for applying design science research*, 2021. arXiv: 2012.04966 [cs.SE].

[38] Kent Beck, Mike Beedle, Arie van Bennekum. "Agile manifesto," Agile Alliance. (2001), [Online]. Available: http://agilemanifesto.org/ (visited on 03/27/2024).

[39] A. Q. Jiang, A. Sablayrolles, A. Mensch, *et al.*, *Mistral 7b*, 2023. arXiv: 2310.06825 [cs.CL].

[40] W. contributors, *Stop word — Wikipedia, the free encyclopedia*, [Online; accessed 29-April-2024], 2024. [Online]. Available: https://en.wikipedia.org/wiki/Stop_word.

A

Appendix 1

A.1 Evaluation Form

You will be presented with a query for retrieving information from the sensory data log file. Accommodating this question are around 30 column names. You will rate if you would be able to use these columns to fulfill the query request.

Columns:
column 1
column 2
etc....

On a scale of 1-5, would you be able to fulfill the request using these columns?
Where 5 means you would be able to fulfill the request fully with conditions.
2-3 you would be able to fulfill the request with some more or less important conditions missing.
1 means some important column is missing and you're unable to create a sufficient query.

score: ___

If any obvious columns are missing, please note them here: "___".

A.2 LLM generated SQL-code

A.2.1 Query set

A.2.2 Design cycle

A.2.3 Refinement cycle

Query
Find where left side emergency lka interventions are triggered
Find where right side drowsiness lka interventions are triggered
Find road edge interventions
Find all cases where the ldw trigger goes high
Get all abort condition signals for lka
Get all trigger condition signals for left side lka
Get all enable condition signals for right side ldw
Get all suppression condition signals for left side ldw
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit changed from 25 meters per second to 15 meters per second?
Did the car ever travel on a motorway faster than 25 meters per second with no detected road work and the reported speed limit spiked to some value that was held for less than four seconds?
Events where driver overrides feature acceleration?

Table A.1: The set of relevant queries provided by Zenseact

```
SELECT * FROM
(
    SELECT
        d.vehicle,
        d.utc_start,
        d.utc_end,
        d.timestamp,
        d."zen_qm_feature_a/lka_lateral_control_path/data/borders/comfort_border/right/type" as
            comfort_border_right,
        d."zen_qm_feature_a/lka_lateral_control_path/data/borders/comfort_border/left/type" as
            comfort_border_left,
        d."zen_qm_feature_a/lka_lateral_control_path/data/borders/hard_border/right/type" as hard_border_right,
        d."zen_qm_feature_a/lka_lateral_control_path/data/borders/hard_border/left/type" as hard_border_left,
        d."zen_qm_feature_a/lss_feature_outputs/data/lka/intervention_info/lateral_border_type" as
            lateral_border_type,
        d."zen_qm_feature_a/lss_feature_output_annotations/data/lka/intervention_side" as intervention_side,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/lateral_border_type" as
            lka_intervention_info_lateral_border_type,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
            road_is_not_narrowing/unitless/value" as lka_status_trigger_conditions_left_non_zero_path,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/additional_lateral_offset/right/mode" as
            additional_lateral_offset_right_mode,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
            distance_to_cross_is_within_limit/unitless/value" as
            lka_status_trigger_conditions_left_non_zero_path_distance_limit,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
            distance_to_cross_is_within_limit/unitless/value" as
            lka_status_trigger_conditions_right_non_zero_path_distance_limit,
        d."zen_qm_feature_a/lka_lateral_control_path/data/lane_reference/heading/radians/value" as
            lane_reference_heading,
        LAG(d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side") OVER (PARTITION BY d.vehicle,
             d.utc_start ORDER BY d.timestamp) as previous_intervention_side
    FROM
        read_parquet('schema_generation/parquets/*.parquet.gzip') as d
) AS tmp
WHERE
    (lateral_border_type = comfort_border_left and previous_intervention_side = 1) OR
    (lateral_border_type = hard_border_left and previous_intervention_side = 1) OR
    (lka_status_trigger_conditions_left_non_zero_path_distance_limit = 1 and previous_intervention_side = 1) OR
    (additional_lateral_offset_right_mode = 1 and previous_intervention_side = 1)
```

Figure A.1: Query set 1-0Shot-3attempts Find where left side emergency lka interventions are triggered

```
SELECT * FROM
(
    SELECT
        d.vehicle,
        d.utc_start,
        d.utc_end,
        d.timestamp,
        d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
        d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
        d."zen_qm_feature_a/zen_qm_feature_a_vehicle_motion_state_data/data/longitudinal_velocity/velocity/
            meters_per_second/value" as lon_v,
        d."zen_qm_sensorfusion_a/zen_qm_sensorfusion_a_vehicle_motion_state_data/data/lateral_velocity/velocity
            /meters_per_second/value" as lat_v,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
            road_is_not_narrowing/unitless/value" as road_not_narrowing_left,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/additional_lateral_offset/left/mode" as
            additional_lateral_offset_left,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/lateral_border_type" as
            intervention_info_lateral_border_type_left,
        LAG(intervention_info_lateral_border_type_left) OVER (PARTITION BY d.vehicle, d.utc_start ORDER BY d.
            timestamp) as previous_intervention_info_lateral_border_type_left
    FROM
        read_parquet('schema_generation/parquets/*.parquet.gzip') as d
) AS tmp
WHERE
    (additional_lateral_offset_left = 2 or road_not_narrowing_left = 1) and
         previous_intervention_info_lateral_border_type_left != intervention_info_lateral_border_type_left
    and intervention_info_lateral_border_type_left = 1;
```

Figure A.2: Query set 1-0Shot-3attempts Find where left side emergency lka interventions are triggered

```
SELECT *
FROM (
    SELECT
        vehicle,
        utc_start,
        utc_end,
        timestamp,
        "zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
            distance_to_cross_is_within_limit/unitless/value" AS distance_to_cross,
        LAG("zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
            distance_to_cross_is_within_limit/unitless/value") OVER (PARTITION BY vehicle ORDER BY timestamp)
            AS previous_distance_to_cross,
        "zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side" AS lka_intervention_side,
        LAG("zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side") OVER (PARTITION BY vehicle
            ORDER BY timestamp) AS previous_lka_intervention_side
    FROM
        read_parquet('schema_generation/parquets/*.parquet.gzip')
) AS subquery
WHERE
    (lka_intervention_side = 1 AND previous_lka_intervention_side = 0) OR
    (lka_intervention_side = 1 AND (distance_to_cross IS NULL OR previous_distance_to_cross IS NULL))
ORDER BY
    vehicle,
    timestamp;
```

Figure A.3: Query set 1-0Shot-3attempts Find where left side emergency lka interventions are triggered

```
SELECT
    d.timestamp,
    d.vehicle,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side",
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        path_is_strong_enough/unitless/value" AS path_is_strong_enough,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        distance_to_cross_is_within_limit/unitless/value" AS distance_to_cross_is_within_limit,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        road_is_not_narrowing/unitless/value" AS road_is_not_narrowing,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/additional_lateral_offset/right/mode" AS
        additional_lateral_offset_right_mode,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/additional_lateral_offset/left/mode" AS
        additional_lateral_offset_left_mode
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') AS d
WHERE
    d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side" = 2
    AND (
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
            path_is_strong_enough/unitless/value" = 1
        OR d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
            distance_to_cross_is_within_limit/unitless/value" = 1
        OR d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
            road_is_not_narrowing/unitless/value" = 1
    )
    AND d."zen_qm_feature_a/lss_diagnostics/data/lka/additional_lateral_offset/right/mode" IS NOT NULL;
```

Figure A.4: Query set 1-0Shot-3attempts Find where right side drowsiness lka interventions are triggered

```
SELECT
    d.vehicle,
    d.timestamp,
    d."zen_qm_feature_a/lka_lateral_control_path/data/borders/comfort_border/right/type" as
        comfort_border_right,
    d."zen_qm_feature_a/lka_lateral_control_path/data/borders/hard_border/right/type" as hard_border_right,
    d."zen_qm_feature_a/lss_feature_outputs/data/lka/intervention_info/lateral_border_type" as
        intervention_lateral_border_type,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        distance_to_cross_is_within_limit/unitless/value" as non_zero_path_distance_within_limit,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        road_is_not_narrowing/unitless/value" as road_not_narrowing,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        path_is_strong_enough/unitless/value" as path_is_strong_enough,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/additional_lateral_offset/right/mode" as
        additional_lateral_offset_right,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/lateral_border_type" as
        intervention_lateral_border_type_annotated,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_drowsiness/right/unitless/value" as
        drowsiness_enabled_right,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status/right/unitless/value" as enabled_right,
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') as d
WHERE
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        distance_to_cross_is_within_limit/unitless/value" = 1
    AND d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_drowsiness/right/unitless/value" = 1
    AND d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status/right/unitless/value" = 1;
```

Figure A.5: Query set 1-0Shot-3attempts Find where right side drowsiness lka interventions are triggered

```
SELECT
    d.vehicle,
    d.utc_start,
    d.utc_end,
    d.timestamp,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        distance_to_cross_is_within_limit/unitless/value" AS distance_to_cross_is_within_limit,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/additional_lateral_offset/right/mode" AS
        lka_additional_lateral_offset_mode
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') AS d
WHERE
    d."zen_qm_feature_a/lss_diagnostics/data/lka/additional_lateral_offset/right/mode" IS NOT NULL
    AND d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        distance_to_cross_is_within_limit/unitless/value" > 0;
```

Figure A.6: Query set 1-0Shot-3attempts Find where right side drowsiness lka interventions are triggered

```
SELECT
    "vehicle",
    "timestamp",
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" AS
        ldw_trigger_status
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" > 0;
```

Figure A.7: Query set 1-0Shot-3attempts Find all cases where the ldw trigger goes high

```
SELECT
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value"
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" = 1;
```

Figure A.8: Query set 1-0Shot-3attempts Find all cases where the ldw trigger goes high

```
SELECT *
FROM (
    SELECT
        d.*,
        LAG("zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value") OVER (ORDER
            BY timestamp) AS prev_ldw_trigger
    FROM
        read_parquet('schema_generation/parquets/*.parquet.gzip') d
) AS subquery
WHERE
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" > 0
    AND (prev_ldw_trigger IS NULL OR prev_ldw_trigger = 0);
```

Figure A.9: Query set 1-0Shot-3attempts Find all cases where the ldw trigger goes high

```
SELECT *
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
  "zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/road_is_not_narrowing
      /unitless/value" = 1;
```

Figure A.10: Query set 1-1Shot-3attempts Find where left side emergency lka interventions are triggered

```
SELECT * FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side" = 0;
```

Figure A.11: Query set 1-1Shot-3attempts Find where left side emergency lka interventions are triggered

```
SELECT
    *
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
    "zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_emergency/left/unitless/value" = 1;
```

Figure A.12: Query set 1-1Shot-3attempts Find where left side emergency lka interventions are triggered

```
SELECT
    *
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
    "zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/zero_path/overshoot_reference/
         unitless/value" = 1
    AND "zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/zero_path/
         valid_trajectory_found/unitless/value" = 1;
```

Figure A.13: Query set 1-1Shot-3attempts Find where right side drowsiness lka interventions are triggered

```
SELECT *
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
     distance_to_cross_is_within_limit/unitless/value" = 1;
```

Figure A.14: Query set 1-1Shot-3attempts Find where right side drowsiness lka interventions are triggered

```
SELECT
    s."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side" AS lka_intervention_side,
    s."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
         distance_to_cross_is_within_limit/unitless/value" AS is_within_limit,
    s."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
         road_is_not_narrowing/unitless/value" AS road_not_narrowing,
    s."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
         distance_to_cross_is_within_limit/unitless/value" AS left_is_within_limit,
    s."zen_qm_feature_a/lss_diagnostics/data/lka/additional_lateral_offset/right/mode" AS
         additional_lateral_offset_right_mode,
    s."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
         path_is_strong_enough/unitless/value" AS path_is_strong_enough
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') s
WHERE
    s."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side" = 2;
```

Figure A.15: Query set 1-1Shot-3attempts Find where right side drowsiness lka interventions are triggered

```
SELECT *
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/left/unitless/value" > 0
   OR "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" > 0;
```

Figure A.16: Query set 1-1Shot-3attempts Find all cases where the ldw trigger goes high

```
SELECT
    *,
    CASE WHEN (
        "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/conditions/right/on_lane_marker/unitless/
            value" > 0
        OR "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/conditions/left/on_lane_marker/unitless/
            value" > 0
        OR "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/left/unitless/value" > 0
        OR "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" > 0
    ) THEN
    TRUE ELSE FALSE END AS ldw_trigger_high
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
    ldw_trigger_high = TRUE;
```

Figure A.17: Query set 1-1Shot-3attempts Find all cases where the ldw trigger goes high

```
SELECT
    *
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" > 0;
```

Figure A.18: Query set 1-1Shot-3attempts Find all cases where the ldw trigger goes high

```
SELECT * FROM
(
    SELECT
        d.vehicle,
        d.utc_start,
        d.utc_end,
        d.timestamp,
        d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
        d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
        d."zen_qm_feature_a/zen_qm_feature_a_vehicle_motion_state_data/data/longitudinal_velocity/velocity/
            meters_per_second/value" as lon_v,
        d."zen_qm_sensorfusion_a/zen_qm_sensorfusion_a_vehicle_motion_state_data/data/lateral_velocity/velocity
            /meters_per_second/value" as lat_v,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side" as lka_intervention_side,
        LAG(lka_intervention_side) OVER (PARTITION BY d.vehicle, d.utc_start ORDER BY d.timestamp) as
            previous_lka_intervention_side,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
            road_is_not_narrowing/unitless/value" as road_is_not_narrowing,
        d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
            distance_to_cross_is_within_limit/unitless/value" as distance_to_cross_is_within_limit,
    FROM
        read_parquet('schema_generation/parquets/*.parquet.gzip') as d
) AS tmp
WHERE
    (lka_intervention_side = 1 and previous_lka_intervention_side is NULL and (road_is_not_narrowing = 1 or
        distance_to_cross_is_within_limit = 1))
```

Figure A.19: Query set 1-2shot-3attempts Find where left side emergency lka interventions are triggered

```
SELECT * FROM
(
    SELECT
        d.timestamp,
        d.vehicle,
        d.utc_start,
        d.utc_end,
        d."zen_qm_sensorfusion_a/zen_qm_sensorfusion_a_vehicle_motion_state_data/data/longitudinal_velocity/
            velocity/meters_per_second/value" as lon_v,
        d."zen_qm_sensorfusion_a/zen_qm_sensorfusion_a_vehicle_motion_state_data/data/lateral_velocity/velocity
            /meters_per_second/value" as lat_v,
        d."zen_qm_feature_a/lss_feature_outputs/data/lka/intervention_info/side" as lka_intervention_side,
        LAG(lka_intervention_side) OVER (PARTITION BY d.vehicle, d.utc_start ORDER BY d.timestamp) as
            previous_lka_intervention_side
    FROM
        read_parquet('schema_generation/parquets/*.parquet.gzip') as d
) AS tmp
WHERE
    (lka_intervention_side = 1 and previous_lka_intervention_side = 0)
```

Figure A.20: Query set 1-2shot-3attempts Find where left side emergency lka interventions are triggered

```
SELECT
    vehicle,
    utc_start,
    utc_end,
    timestamp,
    "zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
    "zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
    "zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_emergency/left/unitless/value" as
        left_emergency_status,
    "zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_emergency/right/unitless/value" as
        right_emergency_status
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
    left_emergency_status = 1 OR right_emergency_status = 1;
```

Figure A.21: Query set 1-2shot-3attempts Find where left side emergency lka interventions are triggered

```
SELECT * FROM read_parquet('schema_generation/parquets/*.parquet.gzip') WHERE "zen_qm_feature_a/lss_diagnostics
    /data/lka/intervention_info/side" = 2;
```

Figure A.22: Query set 1-2shot-3attempts Find where right side drowsiness lka interventions are triggered

```
SELECT
    vehicle,
    utc_start,
    utc_end,
    timestamp,
    lon,
    lat,
    lon_v,
    lat_v,
    lka_intervention_side,
    previous_lka_intervention_side,
    drowsiness_enabled,
    enabled
FROM
    (
        SELECT
            d.vehicle,
            d.utc_start,
            d.utc_end,
            d.timestamp,
            d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) AS lon,
            d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) AS lat,
            d."zen_qm_feature_a/zen_qm_feature_a_vehicle_motion_state_data/data/longitudinal_velocity/velocity/
                meters_per_second/value" AS lon_v,
            d."zen_qm_sensorfusion_a/zen_qm_sensorfusion_a_vehicle_motion_state_data/data/lateral_velocity/
                velocity/meters_per_second/value" AS lat_v,
            d."zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side" AS lka_intervention_side,
            LAG(lka_intervention_side) OVER (PARTITION BY d.vehicle, d.utc_start ORDER BY d.timestamp) AS
                previous_lka_intervention_side,
            d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_drowsiness/right/unitless/value"
                as drowsiness_enabled,
            d."zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status/right/unitless/value" as enabled,
        FROM
            read_parquet('schema_generation/parquets/*.parquet.gzip') as d
    ) AS tmp
WHERE
    (drowsiness_enabled = 1 and enabled = 1 and lka_intervention_side = 2 and previous_lka_intervention_side =
        0)
    OR
    (drowsiness_enabled = 0 and enabled = 0 and lka_intervention_side = 2 and previous_lka_intervention_side =
        0)
ORDER BY
    timestamp;
```

Figure A.23: Query set 1-2shot-3attempts Find where right side drowsiness lka interventions are triggered

```
SELECT *
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE ("zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side") IS NOT NULL
AND ("zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/lateral_border_type") IS NOT NULL
AND ("zen_qm_feature_a/lss_diagnostics/data/lka/intervention_info/side") > 0;
```

Figure A.24: Query set 1-2shot-3attempts Find where right side drowsiness lka interventions are triggered

```
SELECT
    d.vehicle,
    d.utc_start,
    d.utc_end,
    d.timestamp,
    d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
    d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
    d."zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" as
        ldw_status_trigger_right,
    d."zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/left/unitless/value" as
        ldw_status_trigger_left
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') as d
WHERE
    (ldw_status_trigger_right = 1 OR ldw_status_trigger_left = 1)
    AND (ldw_status_trigger_right IS NOT NULL OR ldw_status_trigger_left IS NOT NULL)
```

Figure A.25: Query set 1-2shot-3attempts Find all cases where the ldw trigger goes high

```
SELECT *
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" > 0;
```

Figure A.26: Query set 1-2shot-3attempts Find all cases where the ldw trigger goes high

```
SELECT
    *
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" > 0;
```

Figure A.27: Query set 1-2shot-3attempts Find all cases where the ldw trigger goes high

```
SELECT  *
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE json_extract_scalar(read_json_each(row_to_json(row), json_array_elements_text(row)), '\$.zen_qm_feature_a
     .lss_diagnostics.data.lka.status.trigger.conditions.left.emergency_lane_support.enable_by_cruising_state.
     unitless.value', true) = 1;

Exception: Catalog Error: Scalar Function with name json\_extract\_scalar does not exist!
Did you mean "json\_extract"?
```

Figure A.28: Erroneous: Query set 1-0Shot-2attempts-ToCheckForErrors Find where left side emergency lka interventions are triggered

```
SELECT  COUNT(*)
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/emergency_lane_support/
     enable_by_cruising_state/unitless/value" > 0
AND "zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/emergency_lane_support/
     driver_inattentive/unitless/value" > 0;

Exception: Binder Error: Referenced column "zen\_qm\_feature\_a/lss\_diagnostics/data/lka/status/trigger/
     ↪ conditions/left/emergency\_lane\_support/enable\_by\_cruising\_state/unitless/value" not found in FROM
     ↪ clause!
Candidate bindings: "read\_parquet.zen\_qm\_feature\_a/lss\_diagnostics/data/lka/status/enable/conditions/left/
     ↪ emergency\_lane\_support/enable\_by\_cruising\_state/unitless/value"
```

Figure A.29: Erroneous: Query set 1-1Shot-2attempts-ToCheckForErrors Find where left side emergency lka interventions are triggered

```
SELECT  COUNT(zen_qm_feature_a || 'lss_diagnostics' || '/data/lka/status/trigger/conditions/left/non_zero_path/
     distance_to_cross_is_within_limit/unitless/value') AS emergency_lka_trigger_count
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE (zen_qm_feature_a || 'lss_diagnostics' || '/data/lka/status/trigger/conditions/left/non_zero_path/
     distance_to_cross_is_within_limit/unitless/value') > 0;

Exception: Binder Error: Referenced column "zen\_qm\_feature\_a" not found in FROM clause!
Candidate bindings: "read\_parquet.timestamp"
```

Figure A.30: Erroneous: Query set 1-2Shot-2attempts-ToCheckForErrors Find where left side emergency lka interventions are triggered

```
SELECT  *
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "zen_qm_feature_a/lss_feature_output_annotations/data/lka/status/trigger/conditions/right/
     drowsiness_lane_support/enable_by_cruising_state/unitless/value" IS NOT NULL;


Exception: Binder Error: Referenced column "zen\_qm\_feature\_a/lss\_feature\_output\_annotations/data/lka/
     ↪ status/trigger/conditions/right/drowsiness\_lane\_support/enable\_by\_cruising\_state/unitless/value"
     ↪ not found in FROM clause!
Candidate bindings: "read\_parquet.zen\_qm\_feature\_a/lss\_feature\_output\_annotations/data/lka/status/enable
     ↪ /conditions/right/drowsiness\_lane\_support/enable\_by\_cruising\_state/unitless/value"
```

Figure A.31: Erroneous: Query set 1-0Shot-2attempts-ToCheckForErrors Find where right side drowsiness lka interventions are triggered

```
SELECT  * FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "read_parquet.zen_qm_feature_a/lss_feature_output_annotations/data/lka/status/enable/conditions/right/
     drowsiness_lane_support/enable_by_cruising_state/unitless/value" = 1;


Exception: Binder Error: Referenced column "read\_parquet.zen\_qm\_feature\_a/lss\_feature\_output\_annotations
     ↪ /data/lka/status/enable/conditions/right/drowsiness\_lane\_support/enable\_by\_cruising\_state/unitless
     ↪ /value" not found in FROM clause!
Candidate bindings: "read\_parquet.zen\_qm\_feature\_a/lss\_feature\_output\_annotations/data/lka/status/enable
     ↪ /conditions/right/drowsiness\_lane\_support/enable\_by\_cruising\_state/unitless/value"
```

Figure A.32: Erroneous: Query set 1-1Shot-2attempts-ToCheckForErrors Find where right side drowsiness lka interventions are triggered

```
I apologize for the confusion. The "/" character is not recognized in the SQL syntax. The correct syntax is to
     use double quotes around column names that contain spaces or special characters. Here is the correct SQL
     query:


Exception: Parser Error: syntax error at or near "I"
LINE 2: I apologize for the confusion. The "/"...
```

Figure A.33: Erroneous: Query set 1-2Shot-2attempts-ToCheckForErrors Find where right side drowsiness lka interventions are triggered

```
SELECT  *
FROM json_each_record(read_parquet('schema_generation/parquets/*.parquet.gzip'))
WHERE value->>'zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/left/unitless/value' > '0'
   OR value->>'zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value' > '0';


Exception: Catalog Error: Table Function with name json\_each\_record does not exist!
Did you mean "st\_read"?
```

Figure A.34: Erroneous: Query set 1-0Shot-2attempts-ToCheckForErrors Find all cases where the ldw trigger goes high

```
SELECT  *
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "zen_qm_feature_a_right_trigger_status" = 1
   OR "zen_qm_feature_a_left_trigger_status" = 1;


Exception: Binder Error: Referenced column "zen\_qm\_feature\_a\_right\_trigger\_status" not found in FROM
     ↪ clause!
Candidate bindings: "read\_parquet.zen\_qm\_feature\_a/cruising\_state/data"
```

Figure A.35: Erroneous: Query set 1-1Shot-2attempts-ToCheckForErrors Find all cases where the ldw trigger goes high

```
SELECT  *
FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE zen_qm_feature_a/lss_diagnostics/data/ldw/status/enable/conditions/right/traffic_assist_support/
     driver_drowsiness/unitless/value > 0
AND zen_qm_feature_a/lss_diagnostics/data/ldw/status/enable/conditions/left/traffic_assist_support/core_enable/
     unitless/value > 0
AND zen_qm_feature_a/lss_diagnostics/data/ldw/status/enable/conditions/right/drowsiness_lane_support/
     driver_settings/unitless/value > 0
AND zen_qm_feature_a/lss_diagnostics/data/ldw/status/enable/conditions/left/traffic_assist_support/core_enable/
     unitless/value > 0;


Exception: Parser Error: syntax error at or near "/"
LINE 3: ...neration/parquets/*.parquet.gzip')
WHERE zen\_qm\_feature\_a/lss\_diagnostics/dat...
```

Figure A.36: Erroneous: Query set 1-2hot-2attempts-ToCheckForErrors Find all cases where the ldw trigger goes high

```
SELECT
    d.vehicle,
    d.utc_start,
    d.utc_end,
    d.timestamp,
    d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
    d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
    d."zen_qm_feature_a/lss_feature_outputs/data/lka/intervention_info/lateral_border_type" as
         lka_intervention_side,
    d."zen_qm_feature_a/lka_lateral_control_path/data/borders/comfort_border/left/type" as
         comfort_border_left_side,
    d."zen_qm_feature_a/lka_lateral_control_path/data/borders/hard_border/left/type" as hard_border_left_side,
    d."zen_qm_feature_a/lka_lateral_control_path/data/lane_reference/heading/radians/value" as heading
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') as d
WHERE
    (lka_intervention_side = 1 and (comfort_border_left_side = 1 or hard_border_left_side = 1))
```

Figure A.37: Query set 1-2Shot-1attempts after refinement cycle, Find where left side emergency lka interventions are triggered

```
SELECT * FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
     distance_to_cross_is_within_limit/unitless/value" > 0;
```

Figure A.38: Query set 1-2Shot-1attempts after refinement cycle, Find where left side emergency lka interventions are triggered

```
SELECT
    d.vehicle,
    d.utc_start,
    d.utc_end,
    d.timestamp,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
         road_is_not_narrowing/unitless/value" as road_not_narrowing,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
         distance_to_cross_is_within_limit/unitless/value" as distance_within_limit
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') as d
WHERE
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
         road_is_not_narrowing/unitless/value" != 0
    AND d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/left/non_zero_path/
         distance_to_cross_is_within_limit/unitless/value" != 0;
```

Figure A.39: Query set 1-2Shot-1attempts after refinement cycle, Find where left side emergency lka interventions are triggered

```
SELECT
    d.vehicle,
    d.utc_start,
    d.utc_end,
    d.timestamp,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        distance_to_cross_is_within_limit/unitless/value" as right_distance_to_cross_is_within_limit,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        road_is_not_narrowing/unitless/value" as right_road_is_not_narrowing,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        path_is_strong_enough/unitless/value" as right_path_is_strong_enough
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') as d
WHERE
    (right_distance_to_cross_is_within_limit > 0 AND right_road_is_not_narrowing > 0 AND
        right_path_is_strong_enough > 0)
    OR (right_distance_to_cross_is_within_limit = 0 AND right_road_is_not_narrowing > 0 AND
        right_path_is_strong_enough > 0)
    OR (right_distance_to_cross_is_within_limit = 0 AND right_road_is_not_narrowing = 0 AND
        right_path_is_strong_enough > 0)
    OR (right_distance_to_cross_is_within_limit = 0 AND right_road_is_not_narrowing = 0 AND
        right_path_is_strong_enough = 0)
```

Figure A.40: Query set 1-2Shot-1attempts after refinement cycle, Find where right side drowsiness lka interventions are triggered

```
SELECT
    d.vehicle,
    d.utc_start,
    d.utc_end,
    d.timestamp,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        distance_to_cross_is_within_limit/unitless/value" AS non_zero_path_distance,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        road_is_not_narrowing/unitless/value" AS road_is_not_narrowing,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        path_is_strong_enough/unitless/value" AS path_is_strong_enough
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') d
WHERE
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        distance_to_cross_is_within_limit/unitless/value" > 0
    AND d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        road_is_not_narrowing/unitless/value" > 0
    AND d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        path_is_strong_enough/unitless/value" > 0
```

Figure A.41: Query set 1-2Shot-1attempts after refinement cycle, Find where right side drowsiness lka interventions are triggered

```
SELECT
    d.vehicle,
    d.utc_start,
    d.utc_end,
    d.timestamp,
    d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
    d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        distance_to_cross_is_within_limit/unitless/value" as distance_to_cross_is_within_limit,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        road_is_not_narrowing/unitless/value" as road_is_not_narrowing,
    d."zen_qm_feature_a/lss_diagnostics/data/lka/status/trigger/conditions/right/non_zero_path/
        path_is_strong_enough/unitless/value" as path_is_strong_enough
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') d
WHERE
    (distance_to_cross_is_within_limit > 0 OR road_is_not_narrowing > 0 OR path_is_strong_enough > 0)
```

Figure A.42: Query set 1-2Shot-1attempts after refinement cycle, Find where right side drowsiness lka interventions are triggered

```
SELECT
    d.vehicle,
    d.utc_start,
    d.utc_end,
    d.timestamp,
    d."zen_qm_mapengine/satellite_data/data/longposn/nanodegrees/value"/POWER(10,9) as lon,
    d."zen_qm_mapengine/satellite_data/data/latposn/nanodegrees/value"/POWER(10,9) as lat,
    d."zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/conditions/right/on_lane_marker/unitless/value"
        as ldw_trigger,
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip') as d
WHERE
    ldw_trigger = 1
ORDER BY
    d.timestamp
```

Figure A.43: Query set 1-2Shot-1attempts after refinement cycle, Find all cases where the ldw trigger goes high

```
SELECT * FROM read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE "zen_qm_feature_a/lss_diagnostics/data/lka/status/enable/status_drowsiness/right/unitless/value" > 0;
```

Figure A.44: Query set 1-2Shot-1attempts after refinement cycle, Find all cases where the ldw trigger goes high

```
SELECT
    timestamp,
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/conditions/right/on_lane_marker/unitless/value"
        as right_condition,
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/conditions/left/on_lane_marker/unitless/value" as
        left_condition,
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" as right_status,
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/left/unitless/value" as left_status
FROM
    read_parquet('schema_generation/parquets/*.parquet.gzip')
WHERE
    "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/conditions/right/on_lane_marker/unitless/value" >
        0
    OR "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/conditions/left/on_lane_marker/unitless/value"
        > 0
    OR "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/right/unitless/value" > 0
    OR "zen_qm_feature_a/lss_diagnostics/data/ldw/status/trigger/status/left/unitless/value" > 0;
```

Figure A.45: Query set 1-2Shot-1attempts after refinement cycle, Find all cases where the ldw trigger goes high