POLITECNICO DI TORINO

Master Degree course in Communications and Computer Networks Engineering

Master Degree Thesis

Advanced Natural Language Processing for Translating synthetic natural language to database queries.

Supervisors
Prof. Alessandro Aliberti
Prof. Lorenzo Bottaccioli
Prof. Edoardo Patti

Candidate
Alessandro Tola

Academic Year 2023-2024

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor Prof. Alessandro Aliberti. I am deeply grateful for his guidance, expertise, and support throughout every stage of this research. His insightful feedback, patience, and encouragement have been instrumental in shaping the direction of this thesis.

I would like to express my sincere gratitude to Mattia Tarchini and Felice Cetrone for their assistance during the development stages of this thesis, particularly during the initial and most challenging times. Their guidance and technical expertise have been instrumental in overcoming obstacles and making progress.

I am grateful to my family for encouragement and belief in my abilities. I am thankful to my friends and peers. Their support and friendship have provided much-needed motivation and inspiration throughout this journey.

Abstract

This thesis addresses contemporary challenges in managing extensive datasets, with a specific focus on the transition from traditional relational databases to non-relational databases (NoSQL). The focus is on enhancing the accessibility of NoSQL databases for non-expert users through natural language queries. Recognizing the prevalence of non-relational databases across industries and the imperative for effective natural language interfaces, the primary contributions of this research include the introduction of a synthetic dataset creation method and the utilization of Large Language Models (LLMs) for natural language to NoSQL translation. This decision stems from the recognition of the absence of an existing dataset tailored to the specific requirements of the research. The dataset, created for NL-to-SQL translation incorporates the WikiSQL dataset, leverages Query templates, NL templates, and data augmentation strategies. This method incorporates learnings from established methodologies to guide the process of creating the synthetic dataset effectively addressing challenges related to time and resource constraints inherent in manual pairing. The evaluation indicates that the synthetic dataset is well-structured, diverse, and efficiently optimized for training natural language to NoSQL translation models. The model section outlines the fine-tuning process for LLMs to refine their capabilities and enhance performance in the specific task of translating natural language queries into NoSQL queries. The Supervised Fine-Tuning is done following a Parameter-Efficient Fine-Tuning (PEFT) methodology through QLoRA. Optimal prompt design takes into account user language and database context. The fine-tuning process of LLaMa2 Large Language Model (LLM), demonstrates good improvements in translating natural language queries into NoSQL queries. Comparing the fine-tuned model with the base model reveals significant advancements, with a trade-off observed as the fine-tuned model's generalization capacity slightly decreases, especially for requests deviating significantly from those in the training dataset. Exploring fine-tuning with larger models presents a promising avenue for overall performance improvement, although challenges related to memory constraints and GPU limitations should be addressed. This research aims to contribute valuable insights to the field of natural language interfaces for NoSQL databases, addressing the critical need for a tailored dataset in the early stages of the research and emphasizing potential improvements and the dynamic evolution of natural language interfaces for NoSQL databases.

Contents

Chapter 1

Introduction

In today's interconnected world, where data reigns supreme, effective management and utilization of information have become a critical aspect for businesses, organizations, and even individuals. The backbone of this digital infrastructure lies within databases, sophisticated systems designed to organize, store, and retrieve data efficiently. Among the database technologies, relational databases have long been the traditional choice, offering structured data storage and a standardized querying language known as SQL (Structured Query Language). However, the exponential growth of data in recent years has strained the capabilities of traditional relational databases. As datasets increase in size and complexity, organizations seek alternative solutions that can handle these challenges adeptly. This quest has led to the rise of NoSQL databases, a diverse family of database management systems that diverge from the rigid structures of their relational counterparts.

At the heart of this shift is the recognition that not all data fits neatly into tables and rows. NoSQL databases offer a flexible approach to data modeling, accommodating various types of data structures, including key-value pairs, document-oriented collections, wide-column stores, and graph databases. MongoDB, CouchDB, Cassandra, and Redis are just a few examples of popular NoSQL databases, each tailored to specific use cases and data requirements. The adoption of NoSQL databases brings forth a paradigm shift not only in data storage but also in querying methodologies. Unlike the standardized SQL queries used in relational databases, NoSQL databases employ their own query languages, which vary widely based on the database model and vendor. These query languages often prioritize flexibility and scalability, allowing developers to perform complex operations on semi-structured and unstructured data with ease.

Central to the operation of any database is the ability to query and retrieve information efficiently. Database queries serve as the bridge between users and the underlying data, enabling them to extract meaningful insights and drive informed decision-making. A database query is essentially a request for specific information from a database, typically formulated using a query language supported by the database management system. These queries can range from simple searches for individual records to intricate operations involving multiple data sources and complex conditions. In the context of NoSQL databases, querying takes on a more diverse and nuanced form. For instance, in document-oriented databases like MongoDB, queries often revolve around the structure of JSON (JavaScript Object Notation) documents, leveraging nested fields and array elements for data retrieval.

Despite the versatility and power of NoSQL databases, there remains a barrier for non-technical users to interact with these systems effectively. While developers and data engineers may possess the requisite skills to craft queries in NoSQL query languages, the same cannot be said for business analysts, managers, or domain experts who may lack proficiency in programming or database administration. This disparity underscores the need for a user-friendly interface that can bridge the gap between natural language communication and database querying. By enabling users to formulate queries in everyday language, without the need to deal with the intricacies of query languages or database schemas, organizations can democratize access to data-driven insights and empower decision-makers at all levels.

It is within this context that this research endeavors to make a significant contribution. By focusing on the translation of natural language queries into NoSQL queries, we aim to provide a practical solution for users to interact with non-relational databases intuitively. Our approach seeks to democratize access to data analytics tools and unlock the full potential of NoSQL databases for a broader audience.

One of the fundamental challenges in natural language processing (NLP) tasks, such as NL-to-NoSQL translation, lies in the availability of high-quality training data. Traditional approaches often rely on manually annotated datasets, where human annotators meticulously label examples of natural language queries paired with their corresponding NoSQL equivalents. While effective, this manual annotation process is labor-intensive, time-consuming, and inherently limited in scalability. Recognizing this bottleneck, our research attempts to circumvent the reliance on manual annotation by introducing a novel approach to dataset creation: synthetic dataset generation. By leveraging the principles of data synthesis and generation, we aim to produce a diverse and representative corpus of training examples that capture the breadth and variability of real-world query scenarios.

The utilization of a synthetic dataset holds immense promise for enhancing the fine-tuning process of Large Language Models (LLMs) for NL-to-NoSQL translation. By systematically crafting a diverse array of query examples spanning various query types, database structures, and linguistic patterns, we can effectively expose the LLM to the rich landscape of NoSQL query semantics. This exposure allows the model to learn the intricate mappings between natural language expressions and their corresponding NoSQL representations. Moreover, the synthetic dataset affords us the flexibility to tailor the training data to specific use cases, domain-specific terminology, and query complexities. By enriching the training data with nuanced variations and edge cases that may be underrepresented in conventional datasets, we empower the LLM to generalize more effectively and exhibit greater robustness in handling unseen query patterns and linguistic nuances. In essence, the synthetic dataset serves as a catalyst for enhancing the adaptability, accuracy, and efficacy of the LLM in the domain of NL-to-NoSQL translation, laying the groundwork for more proficient and user-friendly interactions with non-relational databases.

Our main contributions include:

- Synthetic Dataset Creation: Introducing a practical solution for dataset generation by creating a synthetic dataset encompassing a broad range of NoSQL query types. This addresses the limitations of manual pairing and the associated labor-intensive and costly efforts.

- NL-to-NoSQL Translation Fine-tuned LLM: Utilizing Large Language Models (LLMs) for NL-to-NoSQL translation by fine-tuning them on the unique characteristics of non-relational databases. This approach leverages the power of advanced neural networks to improve the accuracy and efficiency of natural language queries conversion into meaningful NoSQL queries.

The thesis is organized as follows. In Chapter 2, we delve into the process of Synthetic Dataset Creation. This exploration takes us through the significance and nuances involved in crafting a synthetic dataset tailored for Natural Language to NoSQL interactions. Chapter 3 is dedicated to the NL-to-NoSQL Translation Model. Here, we detail our proposed methodology for the translation of natural language queries into syntactically and semantically correct NoSQL queries. In Chapter 4 the focus is on the fine-tuning process of Large Language Models (LLMs), where we adapt and optimize these models to address the unique challenges posed by non-relational databases. As we progress to Chapter 5, the spotlight is on the results obtained during the syntehtic dataset creation process and the LLM fine-tuning for the NL-to-NoSQL translation. Through systematic evaluation, we aim to validate the effectiveness and efficiency of our proposed approaches. Chapter 6 marks the conclusion of the thesis and we synthesize the findings, highlight our contributions, and propose avenues for future research.

Chapter 2

Dataset

Recently, there has been a growing interest in neural machine translation (NMT) approaches that formulate the text-to-SQL problem as a language translation problem, and train a neural network on a large amount of NL query/SQL pairs. These approaches have bloomed due to the recent advances in deep learning and natural language processing (NLP), along with the creation of two large datasets (WikiSQL [51] and Spider [48]) for training text-to-SQL systems.

Text-to-SQL maps natural language questions on the given relational database into SQL queries [2,13]. Most previous works [11,16,17,40,49] focus on extracting the question-to-SQL patterns and generalizing them by training an encoder-decoder model with Text-to-SQL corpus. In recent years, large language models (LLMs) have emerged as a new paradigm for Text-to-SQL. Different from prior studies, the core problem in LLM-based Text-to-SQL solution is how to prompt LLM to generate correct SQL queries, namely prompt engineering.

In today's digital age, non-relational databases are used in almost every industry to store information. Non-Structured Query Language (NoSQL) databases are increasingly being used for large-scale data sets, search engines, and real-time web applications. Many organizations are gradually looking into approaches to understand and analyze this enormous unstructured data.

Natural language is a promising alternative interface to DBMSs because it enables non-technical users to formulate complex questions in a more concise manner than SQL or NoSQL. The core problem with existing approaches is that they require an enormous amount of training data in order to provide accurate translations. This training data is extremely expensive to curate, since it generally requires humans to manually annotate natural language examples with the corresponding NoSQL queries (or vice versa). The lack of an existing dataset requires a systematic exploration of methods and previous works to guide the creation of a dataset [6,18,41,43,44] that reflects the special features of natural language queries and their corresponding NoSQL counterparts.

2.1 Synthetic Dataset Construction

Creating a synthetic dataset for Natural Language (NL) requests and NoSQL queries offers a practical solution, especially when faced with the challenges of time and resources involved in manually pairing each request with its corresponding query. Traditionally, generating datasets for NL-to-NoSQL interactions required meticulous manual efforts to match NL requests with suitable NoSQL queries. However, this approach is not only labor-intensive but also costly, limiting the dataset's scale and diversity. Our proposed solution suggests the creation of a synthetic dataset that encompasses a broad range of NoSQL query types.

The lack of an existing public dataset requires a systematic exploration of previous methods to guide the creation of a dataset [6,18,41,43,44] that reflects the special features of natural language queries and their corresponding NoSQL counterparts.

2.1.1 Previous approaches for NL-to-SQL query generation

The paper "Question Generation from SQL Queries Improves Neural Semantic Parsing" [6] attempts to uncover the path to achieving state-of-the-art accuracy in neural semantic parsing with a reduced amount of supervised training data, focusing its investigation on WikiSQL, the largest hand-annotated semantic parsing dataset. At its core, the approach involves a Question Generation Component that takes SQL queries as inputs and generates natural language questions. Leveraging a small-scale supervised training dataset, the generated question-SQL pairs act as pseudo-labeled data, enriching the training of the semantic parser, abbreviated as STAMP (Syntax- and Table-Aware Semantic Parser). The Semantic Parsing Model operates in an end-to-end fashion, with STAMP taking a natural language question as input and generating a SQL query for execution on a table to obtain the answer. The Question Generation Model relies on sequence-to-sequence learning, adopting a copying mechanism to replicate rare words from SQL queries and incorporating latent variables for diversity. Notably, the introduction of question generation proves instrumental in achieving state-of-the-art performance with a reduced amount of supervised data. The paper concludes by suggesting future work involving the incorporation of table information and external knowledge to further enhance the question generation model.

Addressing the challenge of constructing sizable training datasets for natural language to SQL (NL2SQL) translation in database management systems, the "DBPal: A Fully Pluggable NL2SQL Training Pipeline" [43] paper highlights a common hurdle in existing deep learning approaches: the demand for extensive manually curated training data, a time-consuming and costly effort. They propose DBPal, a fully pluggable NL2SQL training pipeline designed to circumvent these challenges. The primary objective is to enhance translation accuracy, fortify the model against linguistic variations, and tailor it for specific databases. DBPal's system architecture is comprehensive, featuring a Generator that uses the database schema and seed templates to generate an initial set of NL-SQL pairs. Augmentation comes into play by leveraging general-purpose data sources and models to linguistically modify the NL part of each pair. A Lemmatizer normalizes the representation of individual words in the resulting NL-SQL pairs. Data instantiation involves

Query Templates using SQL templates to instantiate different possible SQL queries based on the database schema. Corresponding NL Templates are defined for direct translation, using dictionaries of synonymous words and phrases to fill in NL slots. Instances are balanced by randomly sampling from possible instances. The Data Augmentation strategy encompasses automatic paraphrasing using the Paraphrase Database (PPDB) and introducing missing information by randomly dropping words and subphrases from NL training queries. These strategies enhance the model's robustness to varying contexts.

The "Data Augmentation with Hierarchical SQL-to-Question Generation for Cross-domain Text-to-SQL Parsing" [44] paper introduces a straightforward yet effective data augmentation framework that sidesteps the need for human involvement in creating datasets. The proposed Data Augmentation Approach involves SQL Query Generation using Abstract Syntax Tree Grammar (ASTG) to generate SQL queries with varying complexity levels. Hierarchical SQL-to-Question Generation decomposes SQL queries into clauses, translating each clause into a subquestion using a Seq2Seq model with a copy mechanism. The model then assembles these subquestions into a complete natural language question, maintaining the execution order of clauses. The Clause-to-Subquestion Translation Model relies on a standard Seq2Seq model, with SQL tokens represented by word embeddings and token type embeddings. Training Data Construction involves aligning tokens in SQL queries and NL questions, ensuring a seamless mapping. The contributions lie in presenting a resource-cheap data augmentation framework for cross-domain text-to-SQL parsing and proposing a hierarchical SQL-to-question generation model.

2.1.2 DBPal in details

DBPal's [43] innovative approach revolves around a novel training pipeline designed for Natural Language Interfaces to Databases (NLIDBs). It leverages the concept of weak supervision, wherein the database schema becomes a pivotal input for automatically generating a substantial volume of NL-SQL pairs. Unlike traditional methods, DBPal liberates itself from the shackles of manual annotation, offering a more scalable and efficient solution.

System Architecture: DBPal's pipeline unfolds in three key stages: Generator, Augmentation, and Lemmatizer.

- Generator: The initial step involves the Generator utilizing the database schema and seed templates to create an initial training set. These seed templates, covering typical classes of SQL queries, require minimal one-time composition effort and remain independent of the target database. This innovative approach assumes that the database schema provides human-understandable table and attribute names, which are then employed to instantiate the templates.

- Augmentation: A cornerstone of DBPal's efficacy is the Augmentation step. This phase dynamically expands the training data by linguistically modifying the NL part of each pair, ensuring robustness to diverse linguistic variations. Techniques like automatic paraphrasing using external resources add layers of linguistic diversity, enriching the training dataset.

- Lemmatizer: The final stage involves lemmatization, normalizing the representation of individual words in NL-SQL pairs. This process, applied at runtime, simplifies the analysis by mapping different forms of the same word to its root.

Data Instantiation - Crafting Query Templates: The main observation of the instantiation step is that SQL, as opposed to NL, has significantly less expressivity. We therefore use query templates to instantiate different possible SQL queries that a user might phrase against a given database schema, such as ”Select Attribute(s) From Table Where Filter”. For each SQL template, we define one or more NL templates as counterparts for direct translation, such as: ”SelectPhrase Attribute(s) FromPhrase Table(s) WherePhrase Filter”. To account for the expressivity of NL compared to SQL, our templates contain slots for speech variation (e.g., Select-Phrase, FromPhrase, WherePhrase) in addition to slots for database objects (e.g., tables, attributes). Then, to instantiate the initial training set, the Generator repeatedly instantiates each of our NL templates by filling in the corresponding slots. Table, column, and filter slots are filled using information from the database's schema, while a diverse array of NL slots are filled using manually crafted dictionaries of synonymous words and phrases.

Maintaining a balanced distribution of NL-SQL pairs during training data instantiation is crucial. A naive approach, replacing template slots with all possible combinations, risks biasing the model towards templates with more slots. This imbalance can lead to a skewed training set, favoring certain translations due to frequency. To address this, random sampling ensures diverse coverage, preventing dominance by specific templates and maintaining a balanced number of instances per template.

Data Augmentation - Embracing Linguistic Diversity: To address the inherent diversity in expressing the same idea in NL, DBPal employs two key augmentation strategies:

- Automatic Paraphrasing: Leveraging the Paraphrase Database, DBPal generates duplicate NL-SQL pairs by paraphrasing words/subphrases in the NL query. Adjustable parameters govern the extent of paraphrasing, providing a nuanced control over linguistic variations.

- Missing Information Handling: Recognizing the challenge of missing or implicit information, DBPal introduces a strategy to drop words and subphrases from NL training queries. Tunable parameters ensure controlled removal, enhancing the model's resilience to missing context.

2.2 Dataset creation method

In our quest to develop a robust Natural Language to SQL (NL2SQL) translation model, we find inspiration in the DBPal paper's approach. Deep learning approaches for NL2SQL demand substantial amounts of meticulously labeled data, making the manual creation process time-consuming and resource-intensive. DBPal offers a solution through a pipeline that automates the generation of synthetic training data. Data instantiation in DBPal involves the use of Query Templates and NL Templates, which are instantiated based on

SQL templates and corresponding NL templates. The paper underscores the significance of achieving a balanced dataset by employing random sampling from potential instances. This approach ensures that the dataset is both diverse and representative, contributing to more robust and unbiased model training.

The data augmentation strategies employed by DBPal include automatic paraphrasing, using the Paraphrase Database (PPDB) [28] or a pertained paraphrasing model, to generate duplicate NL-SQL pairs, introducing noise by randomly dropping words and sub-phrases from NL training queries. These strategies contribute to a more robust NL2SQL model.

We are in a situation where we lack an existing dataset to kickstart our model. Unlike the other two papers discussed, DBPal's approach aligns with our initial need to bootstrap the translation model without relying on pre-existing datasets. By leveraging DBPal, we can avoid the need to train intermediate models for NL question creation, simplifying our initial steps. Furthermore, DBPal's emphasis on a automated pipeline aligns seamlessly with our objective of efficiently developing the model without the overhead of manual curation or the need for additional training steps. DBPal's methodology offers a practical and scalable solution, setting the stage for the development of a robust translation model.

2.2.1 Implementation

The proposed concept involves leveraging a public dataset, the WikiSQL dataset introduced in the paper "Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning" [51]. In this context, the WikiSQL dataset serves as a valuable resource for extracting SQL statements responsible for creating tables within a database. The extraction process is targeted at obtaining both the names of tables and their associated fields from the SQL statements. Following this extraction, a meticulous cleaning process is applied to refine the collection of table names and fields. This involves removing extraneous words and elements that may not contribute meaningfully to the subsequent steps.

Once a curated collection of table names and fields is obtained, the next phase of the process involves their integration into parametrized NoSQL queries. These parametrized queries serve as versatile templates, encompassing a comprehensive range of potential "find()" requests that a user might formulate. For example, a parametrized query like

```
"find({ $LOG: [{ FIELD1: { $OP1: 'VALUE1' }},
        { FIELD2: 'VALUE2' }] })"
```

could be completed by substituting the general parameters LOG, OP1, FIELD1, VALUE1, FIELD2, and VALUE2. The parameters constituting a collection field will be replaced with those extracted previously from the SQL dataset. A final example could be:

```
"find({ $and: [{ position: { $gt: 3 }},
        { driver: 'Fernando Alonso' }] })".
```

The natural language request, which is also parametrized, will be completed by replacing the parameters with the corresponding fields. For instance, ”What are the COLLECTION where the FIELD1 is OP1 VALUE1 LOG the FIELD2 is VALUE2?” transforms into ”What are the races where the position is greater than 3 and the driver is ’Fernando Alonso’?”

After obtaining the dataset, the next step in enhancing its richness involves the application of augmentation techniques, with a specific focus on paraphrasing. This technique aims to diversify the dataset by generating alternative versions of Natural Language (NL) requests corresponding to the same NoSQL query, thus enriching the dataset with varied expressions of the underlying queries.

To implement the paraphrasing method, we employed ”Parrot”, a framework grounded in the T5 algorithm developed by Google [32]. The choice of Parrot stems from its effectiveness in preserving meaning while introducing variations in sentence structures, aligning with the objective of creating a more comprehensive and diverse dataset.

Key Metrics for Paraphrases:

- Adequacy (Preservation of meaning): Parrot is designed to ensure that the generated paraphrases maintain the original meaning of the NL requests, contributing to the accuracy and coherence of the augmented dataset.
- Fluency (Grammatical correctness): Another critical metric addressed by Parrot is the grammatical correctness of the paraphrased sentences, ensuring that they are linguistically sound and contextually appropriate.
- Diversity (Lexical/Phrasal/Syntactical changes): Parrot introduces variations in lexical choices, phrasing, and sentence structures, enhancing the diversity of the paraphrased sentences. This diversity is crucial for capturing the nuances and breadth of potential user inputs.

Parrot offers flexibility through adjustable parameters for Adequacy, Fluency, and Diversity, allowing customization to align with specific needs. In the context of conversational engines, Parrot primarily focuses on augmenting texts typed into or spoken to conversational interfaces, making it a suitable choice for our NL-to-NoSQL augmentation task. It’s worth noting that the pre-trained model is optimized for text samples of a maximum length of 32, aligning with the typical interaction length in conversational interfaces.

In leveraging the capabilities of Parrot and its adherence to key metrics, the paraphrasing process becomes a valuable tool in expanding the dataset’s expressiveness, ensuring a robust and nuanced representation of user queries for training and evaluating NL-to-NoSQL translation models.

Chapter 3

Model

The challenge of translating natural language queries to NoSQL query equivalents involves not only comprehending the nuances of the input NL query but also constructing a syntactically and semantically correct NoSQL query based on the underlying database schema. NL queries demand a robust understanding of both the user's intent and the intricacies of the database schema to produce accurate and meaningful NoSQL queries.

Drawing inspiration from the most effective models in translating questions into SQL queries [11,16,17,40,49] and from previous works in Natural Language to NoSQL Query Conversion [8,24,36], a significant area to explore involves adapting these methodologies to the domain of NoSQL queries. The foundational basis for this extension of the successes achieved in SQL translation to the diverse landscape of NoSQL databases is rooted in the commonalities of attribute extraction from natural language queries between SQL and NoSQL scenarios. Despite format differences, shared attributes within queries offer an opportunity for strategic model transformation.

3.1 Text to SQL/NoSQL translation

Natural Language Interfaces for Databases (NLIDBs) play a crucial role in facilitating interactions between users and databases by allowing users to express queries in natural language, which are then translated into the corresponding database query language. This process is particularly essential for both SQL and NoSQL databases, although there are differences in their approaches.

SQL databases: NL to SQL translation involves converting natural language queries into Structured Query Language (SQL) statements. SQL is a standardized language used for managing and manipulating relational databases. The translation process typically includes several key steps:

- Semantic Understanding: The system must understand the meaning behind the natural language query, discerning the user's intent in relational database terms.

- Query Structuring: Once the intent is understood, the system structures the query into a valid SQL statement. This involves identifying the specific elements such as SELECT clauses, WHERE conditions, and JOIN operations.

- Database Schema Mapping: NL to SQL translation requires mapping natural language entities to corresponding elements in the database schema. This includes matching natural language terms to table names, column names, and other relational constructs.
- Syntax Generation: The translated query must adhere to the syntax rules of SQL. This involves ensuring that the generated SQL statement is grammatically correct and semantically accurate.

NoSQL databases: the translation process is conceptually similar but adapted to the specific characteristics of NoSQL query languages, which are often less structured than SQL. NoSQL databases support a variety of query languages, such as MongoDB's Query Language for document databases. The translation process involves:

- Intent Recognition: Understanding the user's intent in the context of NoSQL databases, which may involve querying collections of documents.
- Query Formulation: Structuring the natural language query into a valid NoSQL query, considering the flexible and schema-less nature of NoSQL databases.
- Document Structure Mapping: The translation process includes mapping natural language terms to document fields.
- Syntax Adaptation: Adapting the syntax to the specific NoSQL query language, ensuring that the translated query is both syntactically and semantically correct.

In both NL to SQL and NL to NoSQL translation require a robust understanding of the user's intent, although the specific nuances of intent may vary based on the database type. Both processes involve mapping natural language entities to database constructs. In SQL, this includes tables and columns; in NoSQL, it involves documents fields. The translation processes share the commonality of generating queries with correct syntax.

3.2 Different Approaches to text-to-SQL translation

In the realm of Deep Learning for Natural Language (NL) to SQL translation, two primary approaches have garnered attention. Deep learning techniques often employ sequence-to-sequence models, leveraging either recurrent neural networks (RNNs) or transformers. The objective is to capture the sequential nature inherent in both NL questions and NoSQL queries, fostering a holistic understanding of the relationship between the two. Another technique involves the incorporation of attention mechanisms. These mechanisms enhance the model's capability to focus selectively on specific segments of the input sequence. This heightened focus aids in comprehending and translating complex NL structures, contributing to the overall efficacy of the translation process.

For the specific task of Text-to-NoSQL translation, fine-tuning Large Language Models (LLMs) plays a pivotal role. The utilization of pre-trained LLMs, such as GPT-4 or LLaMA, serves as a robust foundation for Text-to-NoSQL endeavors. These models, having undergone training on extensive text corpora, exhibit a comprehensive understanding of natural language, providing a valuable starting point for the translation task. Supervised Fine-tuning the LLMs involves the utilization of task-specific training data to align the behavior of the pre-trained models with the intricacies of NoSQL query generation. This supervised fine-tuning process enhances the LLM's performance, tailoring its capabilities to the nuances of the targeted downstream task in Text-to-NoSQL translation.

3.2.1 Deep Learning for text-to-SQL

Early approaches to Natural Language Interfaces for Databases (NLIDBs) relied on database schema and indexes, extracting SQL queries based on the NL query's keywords and their relationships in the database graph [9, 10, 22, 52]. Parsing-based methods delved into grammatical structures, mapping NL questions to desired SQL queries [12, 15, 29, 42, 47]. In recent years, Neural Machine Translation (NMT) approaches gained prominence, treating text-to-SQL as a language translation task [7, 40, 51]. These leverage vast datasets like WikiSQL and Spider, coupled with advancements in deep learning and natural language processing.

Generating SQL queries using a sequence-based decoder was initially avoided as it could produce syntax, however recent works [19, 34, 45] have changed the landscape by introducing a series of techniques that minimise the possibility of errors by sequence-based decoders. These techniques have made the use of very powerful pre-trained encoder-decoder models [14, 31] a viable and high-performing option, allowing the systems that use them to achieve top performance in both the Spider and WikiSQL benchmarks.

Deep Learning key steps

The first step is **Schema Linking**. It involves uncovering portions of Natural Language Queries (NLQ) that reference database elements. This process is not without challenges, contending with vocabulary disparities, diverse phrasing, and discrepancies in expressing conditions. Schema linking plays a pivotal role in aligning natural language queries with the corresponding database elements. Performing no schema linking is possible too, in fact, while most recent systems incorporate some form of schema linking in their workflow, earlier ones (e.g. Seq2SQL [51], SQLNet [46]) and even some recent ones (e.g. HydraNet [23], T5+PICARD [34], SeaD [45]) simply rely on their neural components to make predictions.

Given that the inputs (NLQ, DB, schema links) are mainly textual, **Natural Language Representation** is responsible for creating an efficient numerical representation that can be accepted by the encoder. The evolution in Natural Language Representation has witnessed a shift from word embeddings to Transformer-based Pre-trained Language Models (PLMs) (T5 [31], BERT [8], BART [14], RoBERTa [21]). This transition is accompanied by an emergent focus on tailoring PLMs for specialized tasks, such as text-to-SQL.

Input Encoding is the process of further structuring the inputs in a format that can be accepted by the encoder. The significance of Input Encoding lies in transforming heterogeneous inputs into a format compatible with neural networks. Finally, **Output Decoding** consists of designing the structure of the predictions that the network will make. Output Decoding is a critical step in generating the final SQL output from the neural network. **Output Refinement** can be applied during the decoding phase in order to reduce the possibility of errors and to achieve better results (PICARD [34]).

3.2.2 Text-to-SQL Empowered by Large Language Models

While previous works concentrated on extracting question-to-SQL patterns, the emergence of LLMs, opened a new paradigm exploring the transformative impact of Large Language Models (LLMs) on the Text-to-SQL task [5,20,33,35,39]. Large Language Models (LLMs) such as GPT-4 [26] and LLaMA [37] have emerged as transformative tools, pre-trained on massive text corpora to perform various natural language tasks. The core challenge in applying LLMs to Text-to-NoSQL is prompt engineering [20, 25], finding the optimal prompt for effective translation. In-context learning [4], a process where LLMs learn from contextual examples during inference, has proven effective, with recent studies emphasizing the significance of including examples for improved performance. Supervised fine-tuning (SFT) further enhances LLMs' Text-to-SQL capabilities by using additional task-specific training data. Question representation [30, 35], in-context learning, and supervised fine-tuning are identified as essential components in large language model-based Text-to-SQL.

Supervised Fine-Tuning for Text-to-SQL

In the realm of Text-to-SQL, where the goal is to automatically translate natural language questions into SQL queries, supervised fine-tuning emerges as a promising avenue for enhancing the performance of Large Language Models (LLMs).

In the context of Text-to-SQL, the objective of supervised fine-tuning is to improve the performance of a given large language model M using a set of training data T. The training data is represented as $T = (q, s, D)$, where q and s denote the natural language question and its corresponding SQL query on database D. The goal is to minimize the empirical loss, employing a loss function (L) that measures the disparity between the generated query and the ground truth query. Similar to question representation, σ is the function that decides the question representation, incorporating relevant information from the schema in the database D. Supervised fine-tuning for Text-to-SQL encompasses two sub-tasks:

- fine-tuning the given LLM M with the supervised data T to obtain the optimal LLM M_* and
- searching for the optimal question representation σ

For the general domain, each item in the supervised data consists of an input prompt p and an expected response r_i from the LLM. To align with the inference process, supervised fine-tuning generates prompt-response pairs from a given Text-to-SQL dataset T. Specifically, the fine-tuning process involves using the target question and the associated database as a prompt $(p_i = \sigma(q_i, D_i))$, treating the desired query as the response $(r_i = s_i)$. Once the data is prepared, existing packages can be employed to perform fine-tuning on the LLM M. This can be achieved through either full fine-tuning or parameter-efficient fine-tuning, depending on the available computational resources. After fine-tuning, the optimized LLM M_* can be utilized for inference, generating queries in response to natural language questions. Importantly, the same question representation σ is used in both the fine-tuning and inference processes.

3.2.3 Using LLMs for Text-to-NoSQL

Translating natural language (NL) questions into database queries, specifically into SQL or NoSQL queries, is a challenging task due to the inherent complexity of human language and the intricate structures of database query languages. In this context, the focus is on the problem of translating NL questions into NoSQL queries, and two primary methodologies stand out: employing Deep Learning techniques or leveraging Large Language Models (LLMs) with fine-tuning for the Text-to-NoSQL task.

Challenges in NL to NoSQL Translation:

- Semantic Ambiguity: Natural language questions often exhibit semantic ambiguity, making it challenging to precisely capture user intent and map it to database query constructs.

- Diverse Query Structures: NoSQL databases, characterized by their schema-less nature, can support a wide variety of query structures. This diversity adds complexity in determining the appropriate structure for a given NL question.

The utilization of Large Language Models (LLMs) for Text-to-NoSQL offers several advantages. Firstly, LLMs possess a broad understanding of natural language as they are pre-trained on extensive text corpora. This foundational knowledge encompasses a wide array of linguistic patterns and structures. Additionally, LLMs reduce the dependency on task-specific feature engineering, thanks to their ability for transfer learning. This adaptability allows them to fine-tune their pre-existing knowledge for specific Text-to-NoSQL tasks with minimal manual engineering efforts, distinguishing them from traditional deep learning models.

Moreover, LLMs exhibit flexibility in handling varied query structures commonly found in NoSQL databases. The schema-less nature of these databases introduces diverse query patterns, and LLMs, with their contextual understanding, can navigate through these without requiring task-specific model modifications. Fine-tuning LLMs specifically for the Text-to-NoSQL task further enhances their performance by tailoring their adaptation to the nuanced intricacies of query generation.

The effectiveness of LLMs in this task is closely tied to the quality of the supervised fine-tuning data. To optimize model performance, it is imperative to ensure that the training dataset is diverse, representative, and aligns with the complexities of the target task.

3.3 Fine-Tuning of LLM

Large language model (LLM) fine-tuning is the process of taking pre-trained models and further training them on smaller, specific datasets to refine their capabilities and improve performance in a particular task or domain. Fine-tuning is about turning general-purpose models and turning them into specialized models. It bridges the gap between generic pre-trained models and the unique requirements of specific applications, ensuring that the language model aligns closely with human expectations. Unlike the pre-training phase, with vast amounts of unstructured text data, fine-tuning is a supervised learning process. This means using a dataset of labeled examples to update the weights of LLM.

Once the instruction dataset is ready we divide the data set into training and test splits. During fine-tuning, we select prompts from the training dataset and pass them to the LLM, which then generates completions. When the model is exposed to a newly labeled dataset specific to the target task, it calculates the error or difference between its predictions and the actual labels. The model then uses this error to adjust its weights, typically via an optimization algorithm like gradient descent. Over multiple iterations (or epochs) of the dataset, the model continues to adjust its weights, honing in on a configuration that minimizes the error for the specific task. The aim is to adapt the previously learned general knowledge to the nuances and specific patterns present in the new dataset, thereby making the model more specialized and effective for the target task. During this process, the model is updated with the labeled data. It changes based on the difference between its guesses and the actual answers. This helps the model learn details found in the labeled data. By doing this, the model improves at the task for which it's fine-tuned.

Parameter-efficient fine-tuning

Training a language model is a computationally intensive task. For a full LLM fine-tuning, we need memory not only to store the model, but also the parameters that are necessary for the training process. Simple hardware cannot handle this amount of hurdle. This is where PEFT is crucial. While full LLM fine-tuning updates every model's weight during the supervised learning process, PEFT methods only update a small set of parameters. This transfer learning technique chooses specific model components and "freezes" the rest of the parameters. The result is logically having a much smaller number of parameters than in the original model (in some cases, just 15-20% of the original weights). This makes memory requirements much more manageable. Not only that, but PEFT is also dealing with catastrophic forgetting. Since it's not touching the original LLM, the model does not forget the previously learned information.

3.3.1 Fine tuning for Text-to-NoSQL

The main idea of creating a model capable of translating a natural language (NL) request into a NoSQL query involves fine-tuning an open-source Large Language Model (LLM). Due to the unaffordable cost of fine-tuning OpenAI LLMs, we focus on open-source LLMs and we choose LLaMA-2-CHAT-7B/13B/70B [38], an up-to-date version of LLaMA

[37]. The fine-tuning was executed using the QLoRA methodology [3]. This approach was selected for its efficiency in reducing memory consumption while maintaining high performance. The process took place on Google Colab, leveraging a T4 GPU.

The primary objective is to determine the optimal prompt for an effective translation, taking into account the user's natural language and the specific context of the database to be queried, defining the information required by the model to generate a semantically correct NoSQL query. It can be also taken into consideration to use specific task-related data examples, aiming to achieve a thorough understanding of the nuances of natural language and the peculiarities of the targeted database. In summary the emphasis is on the optimal design of queries to maximize the effectiveness of the translation process.

Idea of prompt with database information:

```
/* Given the following NoSQL database collection: */
users: id, name, surname, role, company, experience
Answer the following: Which users have the manager role?
```

3.4 Model Evaluation

In the pursuit of a comprehensive and fair evaluation, we adhere to the methodology established in a previous study [50]. This metric quantifies the accuracy of matched keywords between the predicted query and its corresponding ground truth. To rigorously assess the performance of the fine-tuned model, the evaluation of translation correctness was conducted using the BLEU score [27], a widely recognized metric in the field of machine translation (MT). While BLEU is the de facto standard, its reliance on geometric mean calculations of n-gram precisions can occasionally exhibit poor correlation with human judgment at the sentence level. In response to this, we used the smoothing techniques proposed in [1] tailored for enhancing the sentence-level BLEU comparisons.

Chapter 4

Methods

4.1 Dataset Construction

In this section we delve into the specific details of the implementation process of constructing a synthetic dataset for Natural Language (NL) to NoSQL translation, focusing on the key considerations and steps taken during the construction of this synthetic dataset.

Fisrt thing we extracted table names and their corresponding fields from the public dataset ”kaxap/pg-wikiSQL-sql-instructions-80k” available on Hugging Face. This dataset is a converted, cleaned, and syntax-checked version of the SQLWiki dataset, encompassing all SQL statements, including CREATE TABLE statements. From these CREATE TABLE statements, we were able to extract the table names and fields, constructing a collection for the creation of our synthetic dataset.

Create Table Statement
CREATE TABLE ”members” (”name” text, ”birthdate” text, ”role” text, ”characteristics” text);
CREATE TABLE ”tv_ratings” (”episode” text, ”rating” real, ”share” real, ”viewers_millions” real);
CREATE TABLE ”song_list” (”number” real, ”song_title” text, ”lyricist” text, ”singer” text);
CREATE TABLE ”race_calendar” (”race” real, ”circuit” text, ”state” text, ”date” text);

Table 4.1. Some examples SQLWiki dataset’s Create Table Statements

Once this collection was obtained, it underwent a cleansing process to yield table names and fields most suitable for our context, where some words, when taken individually, might lack meaningfulness. To achieve this, words with excessive underscores, those containing attached numbers, or those shorter than 2 characters were eliminated. Subsequently, a manual review was conducted to eliminate any remaining words deemed unsuitable or unclear for our use case. The final collection contains 1800 samples of table names and their corresponding fields.

Collection	Fields
"members"	"name [text], birthdate [text], role [text], characteristics [text]"
"tv_ratings"	"episode [text], rating [real], share [real], viewers_millions [real]"
"song_list"	"number [real], song_title [text], lyricist [text], singer [text]"
"race_calendar"	"race [real], circuit [text], state [text], date [text]"

Table 4.2. Some examples of collections extracted from SQLWiki dataset

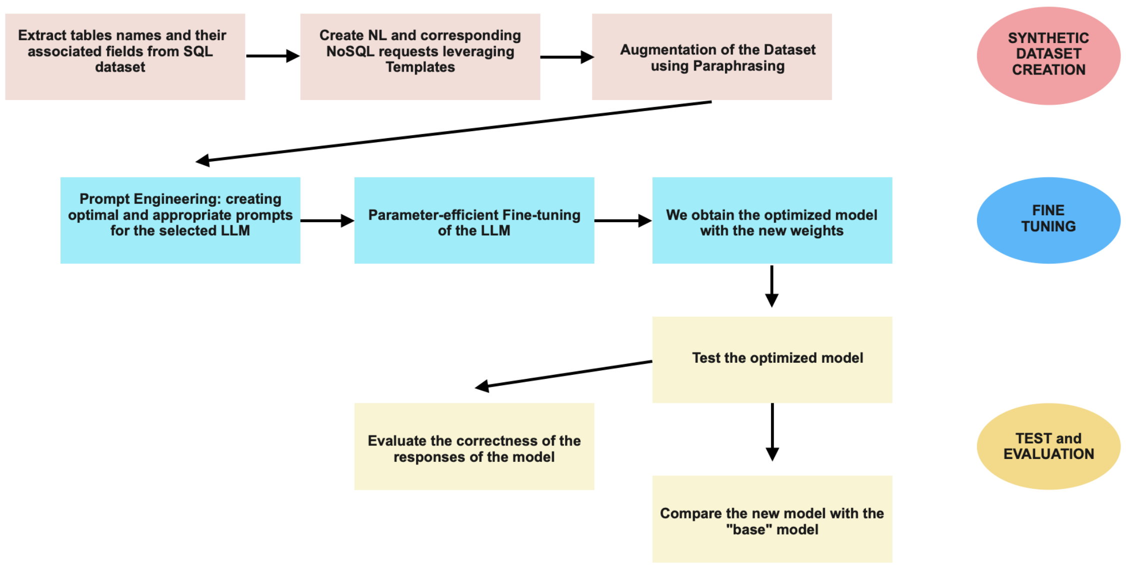
Regarding the templates used to create NL/NoSQL pairs, ten distinct types of "find()" NoSQL queries were formulated. Each query type was pre-coupled with various expressions of the corresponding requests in natural language to already encompass a high number of potential ways to articulate the same query. These templates are fully parameterized concerning request keywords (NL or NoSQL) and the names of fields to which the request pertains. Other parameters were strategically inserted to allow for the modification of the quantity of NL/NoSQL pairs created, ensuring a homogeneous dataset where the number of different query types corresponds to the same number of samples in the dataset.

Choosing LLaMa2 as the reference Large Language Model (LLM), after creating the dataset with NL/NoSQL pairs (retaining the collection name and all its fields in the dataset), the final dataset was constructed for the fine-tuning of the model. This involved generating samples to match the LLaMa2 prompt format. In the realm of prompt engineering, meticulous consideration has been given to construct an optimal prompt that extracts the most accurate and contextually relevant answers from the Large Language Model (LLM). Leveraging the knowledge of the collection name and its associated fields, the prompts are carefully designed to encapsulate the essence of the intended NoSQL query. An illustrative example of such prompt engineering is evident in the following formulation:

Text
"<s>[INST] Given the following structure of a NoSQL collection about members: name, birthdate, role, characteristics, convert the following natural language question into a NoSQL query (the structure of the query should be db.members.find(...)): Select the members where the name is Frank [/INST] find({ name: "Frank" }) </s>"

In this instance, the prompt provides explicit details about the structure of the NoSQL collection, highlighting the fields. By framing the natural language question within the context of the specific collection and its attributes, the prompt ensures that the LLM is equipped with sufficient information to generate a precise and contextually fitting NoSQL query, thereby enhancing the overall effectiveness of the translation process.

Concerning the data augmentation phase, specifically paraphrasing, we employed the Parrot framework. Following the methodology outlined in DBPal [43], we augmented the dataset by generating duplicate Natural Language (NL) to NoSQL pairs. This augmentation process involves the random selection of words or subphrases from the NL query, subsequently paraphrasing them using Parrot. During paraphrasing, words and subphrases within the input NL query are randomly replaced with alternative paraphrases provided by Parrot.


Extract tables names and their associated fields from SQL dataset
Create NL and corresponding NoSQL requests leveraging Templates
Augmentation of the Dataset using Paraphrasing
SYNTHETIC DATASET CREATION
Prompt Engineering: creating optimal and appropriate prompts for the selected LLM
Parameter-efficient Fine-tuning of the LLM
We obtain the optimized model with the new weights
FINE TUNING
Test the optimized model
Evaluate the correctness of the responses of the model
Compare the new model with the "base" model
TEST and EVALUATION

An essential consideration is the level of aggressiveness applied in automatic paraphrasing. To address this, we used two parameters for tuning automatic paraphrasing in alignment with DBPal. The first parameter, ”sizepara”, defines the maximum size of sub-clauses (in terms of the number of words) to be replaced in a given NL query. The second parameter, ”numpara”, establishes the maximum number of paraphrases generated as linguistic variations for each subclause. For instance, setting sizepara = 2 would replace subclauses of size 1 and 2 (i.e., unigrams and bigrams) in the input NL query with paraphrases sourced from Parrot. Furthermore, setting numpara = 3 implies that each unigram and bigram can be replaced by up to 3 paraphrases.

Given the computational expense and time constraints associated with the intensive paraphrasing process, especially when dealing with a substantial volume of samples for augmentation, we propose a simplified data augmentation method. In this approach, we selectively choose 20% to 30% of samples from the original dataset, and through Parrot, generate duplicate Natural Language (NL) to NoSQL pairs. This streamlined strategy aims to strike a balance between computational efficiency and the augmentation of the dataset, providing a pragmatic compromise for datasets of larger scales.

The dataset utilized for fine-tuning the LLama2 Large Language Model (LLM) was curated by incorporating 1000 samples for each type of ”find()” query. To accelerate the paraphrasing process, the simplified approach was employed, selecting 20% of the samples for duplication. This method facilitated an efficient yet robust dataset for the fine-tuning phase, optimizing the model’s performance in NL-to-NoSQL translation.

4.2 Model Fine Tuning

In the process of fine-tuning the LLama2 Large Language Model (LLM) for text-to-NoSQL translation, we adopted an approach that leverages the Quantization through Low-Rank Adaptation (QLoRA) technique [3]. This method stands out for its ability to significantly reduce memory usage during fine-tuning without sacrificing performance.

QLoRA's core methodology involves the implementation of 4-bit quantization, a process that compresses a pre-trained language model by reducing the storage precision of its parameters. The quantized model's parameters are then frozen, and a small number of trainable Low-Rank Adapter layers are introduced. During the subsequent fine-tuning phase, QLoRA strategically backpropagates gradients exclusively through these Low-Rank Adapter layers. This ensures that the frozen 4-bit quantized model remains unaltered, with only the adapters being updated. It's worth noting that the 4-bit quantization employed by QLoRA doesn't compromise the performance of the model; rather, it serves as a memory-efficient alternative to traditional fine-tuning methods.

Contrasting with traditional fine-tuning methods that necessitate updating all model parameters, our approach, known as Parameter-Efficient Fine-Tuning (PEFT), selectively updates a small subset of the model's parameters, optimizing computational efficiency. Supervised fine-tuning (SFT) played a pivotal role, facilitated by the TRL library from HuggingFace, which provides a user-friendly API for creating and training SFT models with minimal code. The SFT Trainer, initialized with relevant parameters including the model, dataset, Lora configuration, tokenizer, and training parameters, facilitated the integration of supervised fine-tuning into the reinforcement learning framework. This meticulous fine-tuning methodology not only optimizes the LLama2 LLM for accurate and efficient Natural Language to NoSQL translation but also enhances accessibility and usability for real-world applications.

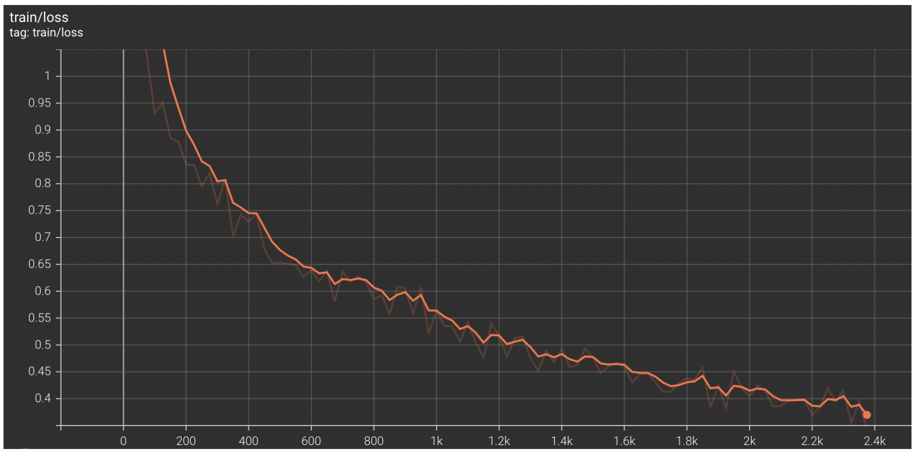


Figure 4.1. This is the loss during the fine tuning.

Chapter 5

Results

5.1 Dataset

The process of constructing a synthetic dataset for fine-tuning the LLama2 Large Language Model (LLM) for Natural Language (NL) to NoSQL translation demonstrated its efficacy addressing the challenges associated with time and resource constraints inherent in manual pairing. Our approach to creating NL/NoSQL pairs was marked by a thoughtful integration of Query Templates, NL Templates, and insights derived from the DBPal framework. This meticulous parameterization, involving request keywords (NL or NoSQL) and field names, ensured the creation of a homogeneous dataset, where the number of different query types matched the samples in the dataset. Following are descriptions of the 10 different "find()" queries that have been considered, along with some samples extracted from the final synthetic dataset that has been created.

- Type 1: Retrieve documents where a field's value is greater than or less than a specified value.
- Type 2: Retrieve documents where a specified field is equal to a given value.
- Type 3: Retrieve documents where a specified field not equal to a given value.
- Type 4: Retrieve documents within a specified range of values for a field.
- Type 5: Retrieve documents where a field exists
- Type 6: Retrieve documents where a field does not exist.
- Type 7: Combine multiple conditions using logical operators like $and and $or.
- Type 8: Retrieve documents where the an array field contains a specific element.
- Type 9: Retrieve documents where an array field matches any of the proposed elements.
- Type 10: Find the document with the maximum / minimum value in a specific field.

Query Type	NL Request	NoSQL Query
Type 1	Select all the douments that have less than 21 posts	find({ posts: { $lt: 21 }})
Type 2	Select the documents where the size is 4	find({ size: 4 })
Type 3	Find the documents that have year not equal to 1994	find({ year: { $ne: '1994' }})
Type 4	Find documents with score values ranging from 25 to 44	find({ score: { $gte: 25, $lte: 44 }})
Type 5	Select documents where the model field exists	find({ model: { $exists: 1 }})
Type 6	Retrieve the documents where the location field is not defined	find({ location: { $exists: 0 }})
Type 7	Show the documents where the minutes are lower than 13 and the city is Miami	find({ $and: [{city: 'Miami'}, {minutes: { $lt: 13} }]})
Type 8	Select records where the names array includes John	find({ names: 'John' })
Type 9	Get documents where the states array includes all the elements found Italy, France and Belgium	find({ states: { $in: ['Italy', 'France', 'Belgium'] }})
Type 10	Take the document with the greater rank value	find().sort({rank:-1}).limit(1)

Table 5.1. Samples from the synthetic dataset

In the realm of prompt engineering, particular attention was given to construct prompts that encapsulate the essence of the intended NoSQL query, leveraging the knowledge of the collection name and its associated fields. The subsequent data augmentation phase, focused on paraphrasing using the Parrot framework, added another layer of richness to our dataset.

In the fine-tuning phase, the dataset, carefully curated with 1000 samples for each type of "find()" query, demonstrated its efficiency in optimizing the LLama2 LLM's performance in NL-to-NoSQL translation. The streamlined approach of selecting 20% of samples for duplication through Parrot struck a pragmatic balance between computational efficiency and dataset augmentation, proving particularly effective for larger-scale datasets.

In summary, the synthetic dataset creation process resulted in a well-structured, diverse, and efficiently optimized dataset. This dataset served as a robust foundation for training models in natural language to NoSQL translation. The successful fine-tuning of the LLama2 LLM showcased the efficacy of our approach, emphasizing its potential for real-world applications in handling NL-to-NoSQL translation challenges.

5.2 Model

The fine-tuning process of the Large Language Model (LLM), specifically LLaMa2-CHAT-7B, exhibited good improvements in translation capabilities for natural language queries into NoSQL queries. Applying a 4-bit quantization technique through QLoRA facilitated efficient fine-tuning on a T4 GPU without compromising performance. By fine-tuning the LLM specifically for the task of NL-to-NoSQL translation, we effectively streamline its focus, enabling it to prioritize the essential task of query conversion without the extraneous information and complexities introduced by the base LLM. This targeted refinement ensures that the model dedicates its cognitive resources solely to understanding the semantic nuances of natural language inputs and generating syntactically correct and contextually relevant NoSQL queries. Consequently, the fine-tuned LLM exhibits improved performance in capturing the subtle intricacies of NoSQL query languages, while also reducing the likelihood of erroneous translations or irrelevant output. In essence, the fine-tuning process serves as a catalyst for optimizing the model's utility in practical applications, facilitating smoother and more intuitive interactions between users and NoSQL databases.

Model	Accuracy (%)	Error Rate (%)
Base Model	65	25
Fine Tuned Model	82	18

The correctness of the model's translations was evaluated using the BLEU score, a widely adopted metric in natural language processing tasks. This evaluation method involved comparing the candidate solution queries generated by the fine-tuned LLM against reference queries, which served as ground truth representations of the intended NoSQL queries. To facilitate this comparison, key words and phrases were extracted from both the reference and candidate queries. For example, from a query such as "find({ age: { $gt: 25} })", the extracted phrase would be "find age gt 25". Subsequently, the similarity between the extracted phrases from the reference and candidate queries was quantified using the BLEU score, which measures the overlap of n-grams between the candidate and reference sentences. A higher BLEU score indicates greater similarity and thus a more accurate translation. This rigorous evaluation approach allowed for a comprehensive assessment of the model's performance in capturing the semantics and structure of natural language requests and effectively converting them into syntactically correct NoSQL queries.

BLEU is not entirely effective but offers several interesting benefits like quick, easy to calculate, language-independent, highly interactive with human interpretation, and widely used.

$$P = \frac{m_t}{w_t}$$

where, m_t is the estimate of tokens from the candidate text that are found in the reference text, and w_t is the total estimate of words in the candidate query. The accuracy is calculated using the equation

$$Accuracy = P \cdot 100\%$$

NL Request	Original Query	Test Query	Accuracy
Retrieve the stadium_and_locations that have Kattie Shields as short_name	find({ short_name: 'Kattie Shields' })	find({ short_name: 'Kattie Shields' })	100
Show the clubs that have 74 matches	find({ matches: { $eq: 74 } })	find({matches: { $gt: 74 }})	75
List the major_awards where the title is not championship or the prize is money	find({ $or: [{ title: { $ne: 'championship' }}, { prize: 'money' }]})	find({ $or: [{ title: { $ne: 'championship' }}, { prize: { $ne: 'money'}}]})	75
Retrieve the from batting_averages where the not_outs is less than 162 or the runs is 726	find({ $or: [{ not_outs: { $lt: 162 }}, { runs: 726 }]})	find({ not_outs: { $lt: 162 }, runs: { $gt: 726 }})	50

Table 5.2. Accuracy of the "Base" Large Language Model on some examples

The BLEU scores obtained for the base model on the test dataset provide a quantitative measure of the model's performance in terms of precision in matching key-words between the predicted NoSQL query and its ground truth. These scores, analyzed collectively, allow us to measure the model's strengths and weaknesses in capturing the nuances of NoSQL queries. These scores indicate a relatively modest level of precision in n-gram matching, suggesting a possible enhancement in translation correctness.

NL Request	Original Query	Test Query	Accuracy
Retrieve the stadium_and_locations that have Kattie Shields as short_name	find({ short_name: 'Kattie Shields' })	find({ short_name: 'Kattie Shields' })	100
Show the clubs that have 74 matches	find({ matches: { $eq: 74 } })	find({matches: { $eq: 74 }})	100
List the major_awards where the title is not championship or the prize is money	find({ $or: [{ title: { $ne: 'championship' }}, { prize: 'money' }]})	find({ $or: [{ title: { $ne: 'championship' }}, { prize: 'money' }]})	100
Retrieve the from batting_averages where the not_outs is less than 162 or the runs is 726	find({ $or: [{ not_outs: { $lt: 162 }}, { runs: 726 }]})	find({ not_outs: { $lt: 162 }, runs: { $eq: 726 }})	75

Table 5.3. Accuracy of the Fine Tuned Large Language Model on some examples

The BLEU scores obtained for the fine-tuned model on the test dataset quantitatively measure the model's precision, indicating a substantial improvement over the base model's scores. Comparing the fine-tuned model with the base model revealed some advancements, however a trade-off was observed, as the model's generalization capacity slightly decreased, especially for requests that deviated significantly from those in the training dataset. Despite this, the improved performance in challenging scenarios positions the fine-tuning LLMs as a promising advancement in the field.

Chapter 6

Conclusion

Concluding this research, several areas for improvement and future considerations emerge, shaping the trajectory of advancements in natural language to NoSQL translation.

Dataset Considerations: The synthetic dataset creation approach has proven valuable, the consideration of employing a larger dataset could significantly enhance the model's training comprehensiveness. The created dataset is well-structured, diverse, and optimized for training natural language to NoSQL translation models. Introducing a broader array of diverse and complex Natural Language variations of the same request, either through the incorporation of additional templates or by increasing the intensity of paraphrasing, has the potential to augment the dataset's richness. This enhancement could contribute to a more robust and nuanced training environment, fostering improved performance and adaptability of the model in handling a wider spectrum of user queries.

Model Evaluation: The fine-tuned model exhibits improved translation capabilities, especially in handling challenging queries. However, there is a trade-off, as the model may lose some generalization capacity, particularly for requests that deviate significantly from those in the training dataset. This nuanced understanding of the model's strengths and limitations underscores the need for ongoing refinement and exploration in the realm of natural language to NoSQL translation.

Future Enhancements: Exploring the fine-tuning of larger models, such as LLama 13B and 70B, presents a promising avenue for improving overall performance. These larger models, with increased parameters and complexity, offer the potential for more nuanced learning. However, this opportunity comes with its own set of challenges, particularly in the context of memory constraints and GPU limitations.

This research lays the groundwork for future endeavors, highlighting possible improvements. The ongoing evolution of natural language interfaces for NoSQL databases remains a dynamic field, poised for continued exploration and innovation.

Bibliography

[1] Boxing Chen and Colin Cherry. A systematic comparison of smoothing techniques for sentence-level BLEU. In Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, and Lucia Specia, editors, *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 2014.

[2] Naihao Deng, Yulong Chen, and Yue Zhang. Recent advances in text-to-SQL: A survey of what we have and what we expect, 2022.

[3] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

[4] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. A survey on in-context learning, 2023.

[5] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. C3: Zero-shot text-to-sql with chatgpt, 2023.

[6] Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. Question generation from SQL queries improves neural semantic parsing, 2018.

[7] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-SQL in cross-domain database with intermediate representation. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[8] Kazi Hossen, Mohammed Uddin, Minhazul Arefin, and Md Ashraf Uddin. Bert model-based natural language to nosql query conversion using deep learning approach. *International Journal of Advanced Computer Science and Applications*, 2023.

[9] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient ir-style keyword search over relational databases, 2003.

[10] Vagelis Hristidis and Yannis Papakonstantinou. Discover: Keyword search in relational databases, 2002.

[11] Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Bowen Li, Jian Sun, and Yongbin Li. S^2sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers, 2022.

[12] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017.

[13] George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql, 2023.

[14] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[15] Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 2014.

[16] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql, 2023.

[17] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing, 2023.

[18] Yuming Li, Rong Zhang, Xiaoyan Yang, Zhenjie Zhang, and Aoying Zhou. Touchstone: Generating enormous Query-Aware test databases, 2018.

[19] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing, 2020.

[20] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability, 2023.

[21] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[22] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. Spark: top-k keyword query in relational databases, 2007.

[23] Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. Hybrid ranking network for text-to-sql, 2020.

[24] Suravi Mondal, Prasenjit Mukherjee, Baisakhi Chakraborty, and Rezaul Bashar. Natural language query to nosql generation using query-response model. *2019 International Conference on Machine Learning and Data Engineering (iCMLDE)*, 2019.

[25] Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies, 2023.

[26] OpenAI. Gpt-4 technical report, 2023.

[27] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin, editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2002.

[28] Ellie Pavlick and Chris Callison-Burch. Simple PPDB: A paraphrase database for simplification, 2016.

[29] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. Modern natural language interfaces to databases: Composing statistical parsing with

semantic tractability. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, 2004.

[30] Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction, 2023.

[31] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

[32] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[33] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-sql capabilities of large language models, 2022.

[34] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[35] Ruoxi Sun, Sercan O. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. Sql-palm: Improved large language model adaptation for text-to-sql, 2023.

[36] Pradeep T, Rafeeque P C, and Reena Murali. Natural language to nosql query conversion using deep learning, 2019.

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, TimothÃ©e Lacroix, Baptiste RoziÃ¨re, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[38] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[39] Immanuel Trummer. Codexdb: Generating code for processing sql queries using gpt-3 codex, 2022.

[40] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers, 2020.

[41] Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. Learning to synthesize data for semantic parsing, 2021.

[42] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. Synthesizing highly expressive sql queries from input-output examples. *SIGPLAN Not.*, 2017.

[43] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin HÃ¤ttasch, Steffen Eger, Ugur Cetintemel, and Carsten Binnig. Dbpal: A fully pluggable nl2sql training pipeline, 2020.

[44] Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. Data augmentation with hierarchical sql-to-question generation for cross-domain text-to-sql parsing, 2022.

[45] Kuan Xu, Yongbo Wang, Yongliang Wang, Zujie Wen, and Yang Dong. Sead: End-to-end text-to-sql generation with schema-aware denoising, 2023.

[46] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning, 2017.

[47] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: query synthesis from natural language. *Proc. ACM Program. Lang.*, 2017.

[48] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2019.

[49] Yanzhao Zheng, Haibin Wang, Baohua Dong, Xingjun Wang, and Changshan Li. Hie-sql: History information enhanced network for context-dependent text-to-sql semantic parsing, 2022.

[50] Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-SQL with distilled test suites. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[51] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017.

[52] Tok Wang Ling Zhong Zeng, Mong Li Lee. Answering keyword queries involving aggregates and group-bys in relational databases, 2015.