Natural Language Processing (2025), 1–32
doi:10.1017/nlp.2024.63

CAMBRIDGE UNIVERSITY PRESS

ARTICLE

DocSpider: a dataset of cross-domain natural language querying for MongoDB

Arif Görkem Özer, Recep Firat Cekinel, Ismail Hakki Toroslu and Pinar Karagoz

Computer Engineering, Middle East Technical University, Ankara, Türkiye
Corresponding author: Arif Görkem Özer; Email: gorkem@ceng.metu.edu.tr

(Received 20 March 2024; revised 23 October 2024; accepted 10 December 2024)

Abstract

Natural language querying allows users to formulate questions in a natural language without requiring specific knowledge of the database query language. Large language models have been very successful in addressing the text-to-SQL problem, which is about translating given questions in textual form into SQL statements. Document-oriented NoSQL databases are gaining popularity in the era of big data due to their ability to handle vast amounts of semi-structured data and provide advanced querying functionalities. However, studies on text-to-NoSQL systems, particularly on systems targeting document databases, are very scarce. In this study, we utilize large language models to create a cross-domain natural language to document database query dataset, *DocSpider*, leveraging the well-known text-to-SQL challenge dataset Spider. As a document database, we use MongoDB. Furthermore, we conduct experiments to assess the effectiveness of the *DocSpider* dataset to fine-tune a text-to-NoSQL model against a cross-language transfer learning approach, SQL-to-NoSQL, and zero-shot instruction prompting. The experimental results reveal a significant improvement in the execution accuracy of fine-tuned language models when utilizing the DocSpider dataset.

Keywords: natural language querying; text-to-NoSQL; cross-domain transfer learning; large language models; document database

1. Introduction

In recent years, the availability of massive amounts of data has highlighted the importance of extracting meaningful information from these data lakes. Almost every electronic device generates data sourced from sensors, logging, or for archival purposes which is required to be stored in data repositories. Databases serve as middleware between the raw data and the end-users. When users seek to retrieve information from databases, they typically construct queries using database-specific languages, such as SQL for relational databases.

Besides relational databases, the increasing amounts of data and the need for efficient access have made NoSQL databases a popular choice among developers and companies due to their flexible data model, scalability, and high performance. These databases provide developers with flexibility, eliminating the need to predefine a strict tabular schema as required by relational databases. NoSQL databases can scale horizontally, allowing applications to effectively manage high loads. Additionally, NoSQL databases can handle high read/write requests in real time, making them preferable for applications involving large volumes of unstructured or semi-structured data. On the other hand, the well-structured data format, ACID compliance, reliable transactions,

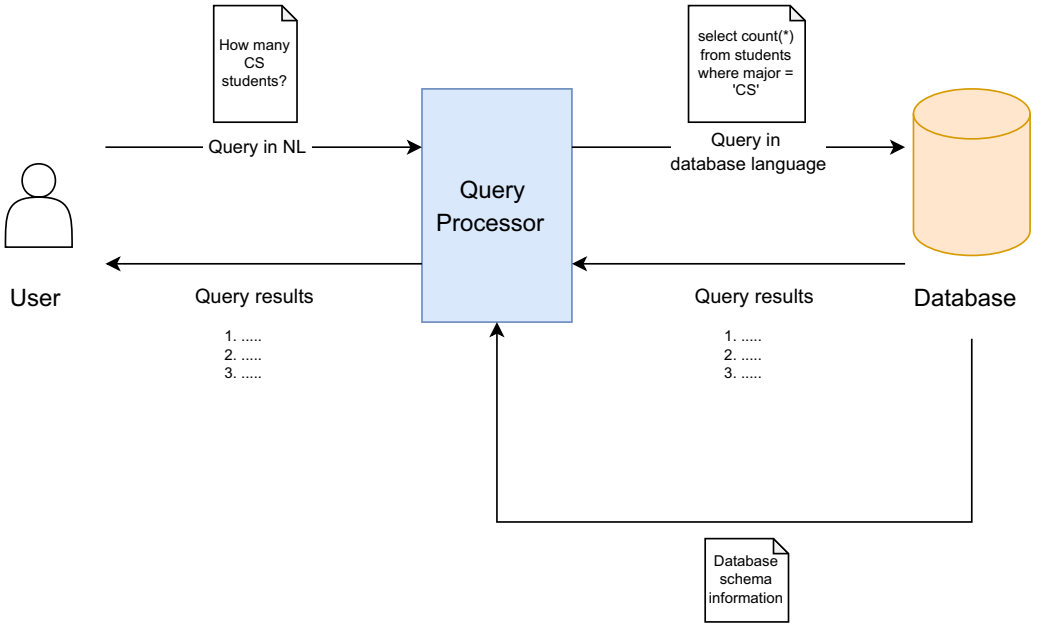

Figure 1. Natural language querying problem.

and data integrity capabilities of relational databases make them more favorable for use cases requiring high data integrity and consistency.

MongoDB[a] (Membrey *et al.* 2010) is a data store for JSON-like documents that supports create, read, update, and delete (CRUD) operations. MongoDB stores objects in BSON format which is a binary encoding of JSON objects. Because of MongoDB's flexible data model and high-performance querying capabilities, it is used for data storage in various applications involving geospatial data (Makris *et al.* 2021), time-series data (Kang *et al.* 2015), or data analytics in general (Mahmood and Risch 2021).

For both relational and NoSQL databases, users are required to have a solid understanding of database management to effectively query the data. The users should know the data type suitable for columns, and the available operations and utilities related to each data type. Despite the fact that these requirements are standard practice for technically proficient users, a significant percent of the individuals face difficulties in direct interaction with data due to such technical barriers.

Natural language querying (NLQ), which is depicted in Figure 1, is a challenging problem. Searching and querying databases using natural language (NL) involves translating a given text statement into an SQL/NoSQL query and executing the translated query against the database, a process often referred to as text-to-SQL/text-to-NoSQL. This paradigm eliminates the need for non-technical users to be familiar with database-specific languages, enabling natural interaction with the database. As stored data increasingly involves multiple modalities (text, tabular, image, etc.), novel systems try to convert NL query structures into database-specific languages (Chen *et al.* 2023).

Early attempts on this task focused on solving this problem for specific database contexts (Owei, Rhee, and Navathe 1997; Li, Yang, and Jagadish 2005; Paparizos *et al.* 2009). Without the existence of language models, such attempts were limited in their ability to handle diverse query structures and they were only useful for certain databases. The subsequent deep learning

[a]https://www.mongodb.com/

and language model solutions such as ELMo (Peters *et al.* 2018) and BERT (Devlin *et al.* 2019) remain limited to provide a diverse NLQ solution, since a cross-domain text-to-SQL dataset was not available at the time, and the models suffer from the generalization problem (Bay and Yearick 2024).

After the introduction of WikiSQL (Bhagavatula, Noraset, and Downey 2015) and Spider (Yu *et al.* 2018) datasets, which include a large collection of NL question–SQL statement pairs for multiple databases, the text-to-SQL solutions had significant advancements. These datasets are still widely employed to train neural models for this task.

In the state-of-the-art text-to-query solutions, the research focus is mostly on translating NL queries to SQL statements. There are only a few NLQ studies targeting NoSQL databases that could be attributed to the scarcity of cross-domain datasets for NoSQL databases. The existing NLQ studies on NoSQL databases generally consider a specific context and does not target handling the generalization problem for neural models (Majeed, Ahmad, and Khalid 2016; Mondal *et al.* 2019). In the literature survey, we have come across only one study challenging the NLQ problem by providing a cross-domain dataset for the graph database Neo4j, to train models for translating text-to-Cypher, the query language for Neo4j (Zhao *et al.* 2023). While Neo4j is the most popular graph database, its popularity is not as high as MongoDB, among the top 20 database management systems (Akhtar 2023).

The main goal of this study is to release a cross-domain text-to-NoSQL dataset, DocSpider, targeting MongoDB. The dataset is generated utilizing the Spider text-to-SQL dataset and translating gold SQL queries[b] to their equivalent MongoDB Query Language (MQL) queries. This transformation is performed using different Large Language Models (LLMs), including state-of-the-art models and also, in particular, those specifically pretrained on coding tasks. The correctness of the translation from SQL to MQL is checked with user studies. Furthermore, the usefulness of the proposed dataset is evaluated with cross-domain and in-context learning experiments. Additionally, the models are fine-tuned with this dataset to verify the dataset's usefulness. Our contribution can be summarized as follows:

- Producing and sharing our DocSpider cross-domain text-to-NoSQL dataset by transforming gold SQL queries in the Spider dataset.
- Proposing a novel approach to translate SQL queries to MongoDB database by utilizing large language models which can be also extended to other NoSQL databases.
- Examining the necessity for our dataset by experimental evaluations[c]

The other sections in this paper are organized as follows: In Section 2, related work about text-to-SQL and text-to-NoSQL tasks are reviewed. In Section 3, methods applied to prepare the dataset are explained. In Section 4, experiments and experiment results are shared, to illustrate the benefits of the introduced dataset. The experiment results are discussed in Section 5. Finally, the paper is concluded in Section 6.

2. Related work

For text-to-SQL and text-to-NoSQL problems, the earlier studies concentrated on rule-based approaches. However, the success of neural models across various tasks shifted attention to deep learning-based models. Nowadays, state-of-the-art methods leverage large language models with prompt engineering.

[b] See Section 3 for explanation of gold SQL.
[c] The GitHub repository can be found at: https://github.com/arifgorkemozer/docspider

2.1 Text-to-SQL

Rule-based methods

As one of the earliest rule-based methods, Owei *et al.* (1997) leveraged conceptual query language to generate SQL statements by identifying source tables, target tables, and relationships in the query composed in natural language. Later on, Kate *et al.* (2018) proposed a rule-based algorithm, which includes tokenization, lexical analysis, syntactic analysis, and semantic analysis. On the other hand, Uma *et al.* (2019) applied part-of-speech (POS) tagging and regular expressions to construct accurate SQL queries. The study used a railway system database. Focusing on a non-English natural language, Jung and Kim (2020) worked on a rule-based solution for SPARQL in Korean language. They aimed to resolve ambiguities (double meanings) in the NL query by looking for all possible queries with the A* algorithm (Dechter and Pearl 1985) between tables.

Deep learning-based methods

For text-to-query translation, Yin *et al.* (2016) developed a Recurrent Neural Network (RNN) based model and used row embeddings to train the model. Each layer of the RNN takes the query embeddings, table row embeddings, and the result of the previous layer as inputs. On the other hand, Zhong *et al.* (2017) developed a Seq2SQL model which uses Long Short Term Memory (LSTM) layers, and they trained the model with reinforcement learning on the WikiSQL (Bhagavatula *et al.* 2015) dataset. Xu *et al.* (2017) presented SQLNet and focused on solving the problem of having different term orders in the input and improved the work in Seq2SQL. Bazaga *et al.* (2021) worked with a transformer model, which was designed to work with multiple tables. They provided as many sentence variations as they could to learn the embeddings for the English language to provide improved performance in text-to-SQL translation.

LLM-based methods

As one of the first LLM-based solutions, Scholak *et al.* (2021) proposed the PICARD method, a text-to-SQL post-processing solution based on T5 (Raffel *et al.* 2020). This method applies constrained beam search on the generated SQLs, allowing the generation of syntactically correct SQL queries. Gao *et al.* (2024) proposed DAIL-SQL, which succeeds in leveraging the in-context learning capacity of LLMs and achieving a balance between performance and token efficiency. Dong *et al.* (2023) developed C3 method, which eliminates the bias for usage of certain SQL clauses, by providing extensive prompts to the ChatGPT model. Rai *et al.* (2023) introduced token preprocessing with semantic parsing and identified compound boundaries to improve the precision of parsing SQL tokens and utilized the T5 model on the Spider dataset.

2.2 Text-to-NoSQL

The text-to-NoSQL studies in the literature cover different NoSQL databases and their query languages (such as document, graph, key-value, and column databases). The use of LLMs is limited; rather template-based and more basic neural architectures are employed.

Majeed *et al.* (2016) designed an automated text-to-XQuery converter utilizing linguistic features such as lemmatization and part-of-speech tagging. Their system identifies keywords, symbols, attributes, values, and relations among different types of queries. Mondal *et al.* (2019) performed syntactic parsing and semantic analysis and then generated MongoDB queries by using a synonym table for table fields.

Blank *et al.* (2019) designed an LSTM (Sak, Senior, and Beaufays 2014) model and utilized GloVe word embeddings (Pennington, Socher, and Manning 2014) to extract entities in the input query. The authors also employed reinforcement learning for training their model and targeted Elasticsearch databases. Hains *et al.* (2019) presented an approach to convert English to the

Cypher querying language, which is used for the Neo4j graph database. Pradeep *et al.* (2019) proposed an LSTM-based solution to the text-to-NoSQL problem, by creating human-annotated dataset for a limited number of question types on the MongoDB database. Abdelhedi *et al.* (2021) proposed an Object2NoSQL approach to convert UML/OCL to NoSQL queries on different kinds of NoSQL databases: *Cassandra (column), MongoDB (document), Neo4j (graph), and Redis (key-value)*. Instead of natural language, they focused on UML queries to provide conversions targeting Big Data.

Hossen *et al.* (2023) developed a BERT-based text-to-MongoDB query conversion system and performed tokenization and part-of-speech tagging as part of the preprocessing. Subsequently, the Levenshtein distance algorithm is used by them to implement the collection and attribute extraction. Then a BERT model is trained to identify the operation in the target MongoDB query. Finally, a syntax tree is used to generate the final query. The authors evaluated their system with WikiSQL dataset. Kobeissi *et al.* (2023) proposed a solution to convert natural language queries to Cypher for the Neo4j database. They detect the intent in the text query and provided a template-based solution to generate Cypher queries. The authors have provided a case study using event logs of a loan application system. Zhang *et al.* (2023a) provided a template-based solution using BART (Lewis *et al.* 2020) for the vaccine adverse events on Elasticsearch database.

For both text-to-SQL and text-to-NoSQL problems, LLM-based models have proven to be successful for query generation. However, as recent state-of-the-art studies have shown, prompt engineering is vital for generating accurate responses from LLMs. Therefore, in this study, these suggestions are incorporated into the solution when designing the prompts for the LLMs. Additionally, we considered models that have demonstrated success when selecting LLMs for query generation.

2.3 Cross-domain datasets

Addressing the NLQ problem in a specific context can be considered easier compared to multi-context domains, owing to the narrower scope. On the other hand, models trained with a cross-domain dataset might perform better on varying domains. In the text-to-SQL scope, the emergence of datasets such as WikiSQL, Spider, CoSQL (Yu *et al.* 2019), BIRD (Li *et al.* 2024), and Archer (Zheng *et al.* 2024a) has provided researchers with opportunities to work on cross-domain datasets.

The Spider dataset stands out for cross-domain model training, as the text–SQL statement pairs are grouped into multiple query complexity levels, which allows researchers to benchmark capability of their models effectively. The query complexity levels were determined according to the number of SQL components and the complexity of the gold query such as whether it is a nested or not. LLMs have demonstrated significant success in the text-to-SQL task within the Spider challenge. Therefore, our study aims to leverage LLMs to release the DocSpider dataset by converting the Spider dataset into text-to-MongoDB queries. We use various LLMs to expand the dataset and examine the performance of different LLM types (e.g., code LLMs vs. base LLMs, or open-source LLMs vs. proprietary LLMs) on this task.

In contrary to text-to-SQL problem, the only cross-domain and context-independent NoSQL dataset in the literature is CySpider. The authors created the dataset from the Spider dataset, and they translated SQL statements to Cypher statements. They implemented an algorithm to map SQL clauses to their corresponding SQL-equivalent Cypher clauses. In this line, our study presented here leverages LLMs to generate MQL queries. Even though we focused on MongoDB, our approach can be applicable to any NoSQL database. Additionally, we performed a user study to assess the consistency of the generated ground-truth queries.

To the best of our knowledge, the only existing studies on manually curating a dataset for the text-to-MongoDB problem are those of Mondal *et al.* (2019) and Pradeep *et al.* (2019). More specifically, the authors evaluated their text-to-MongoDB system using the datasets they

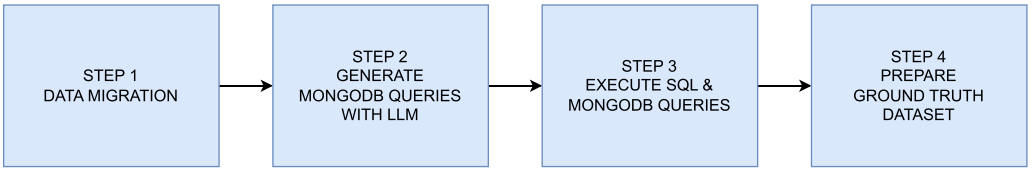

Figure 2. The pipeline overview.

annotated. Unfortunately, the datasets are not publicly available. As a result, DocSpider appears as the first publicly available cross-domain text-to-MongoDB dataset.

3. Preparation of DocSpider dataset

Our reference dataset, Spider,[d] is a benchmark dataset for the text-to-SQL problem, comprising 10,181 questions and 5,693 SQL queries across 200 databases with multiple tables, covering 138 different domains. The main reason for choosing this dataset as input is that it is widely used for text-to-SQL, and that the dataset can be considered realistic since the samples were annotated by humans across diverse databases. At the time of writing this paper, MiniSeek[e] holds the first place on this challenge with 91.2% execution accuracy on the test set.

It is always possible to return the same query results with different SQL queries, and there should be an ideal, non-complex way of querying the database. In the Spider dataset, given an NL query, the corresponding SQL statement provided by human annotators is considered as the *gold-standard SQL*, namely the *gold SQL*. While translating NL queries into MQL queries, gold SQLs from Spider dataset are also used. While solving the text-to-SQL problem, LLM prompts contained database information and the gold SQL to fine-tune (Scholak *et al.* 2021) the models, or to do in-context learning (Gao *et al.* 2024; Dong *et al.* 2023). Hence, we aim to transform the gold SQL queries to their equivalent MQL queries by using LLMs.

According to the Spider leaderboard, LLMs that were successful on coding tasks performed better on SQL generation task. Thus, while selecting the LLMs, we considered those specifically pretrained on coding tasks or demonstrating proficiency in coding. MT-bench (Zheng *et al.* 2023) is a benchmark specifically designed to assess LLMs on different question categories. It consists of 80 questions and evaluates the models' coding, math, reasoning, and conversational flow skills comprehensively. Therefore, we considered language models that achieved satisfactory scores in MT-bench's coding evaluations. The following subsections detail the implementation of the dataset transformation pipeline, which is developed in Python.

Figure 2 and Figure 3 illustrate the overall pipeline of the proposed dataset construction approach. First, the Spider data is transferred to the MongoDB database; the details of the data migration process are provided in Section 3.1. Second, LLMs are employed to generate MQL queries for the given text questions and their corresponding gold SQL pairs. During the MQL query generation, prompts to LLMs are used, including the schema and foreign key relations. The details of SQL-equivalent MQL query generation are presented in Section 3.2. Subsequently, as discussed in Section 3.3, the generated MQL queries and gold SQL queries are executed on databases and the execution results are compared. Based on the execution results, queries that yield similar results with minor differences (such as ordering and projection) are manually manipulated. Finally (see Section 3.4), a subset of the DocSpider dataset is annotated by the reviewers to assess the consistency and the reliability of the dataset.

[d] https://yale-lily.github.io/spider
[e] https://www.seek.ai/

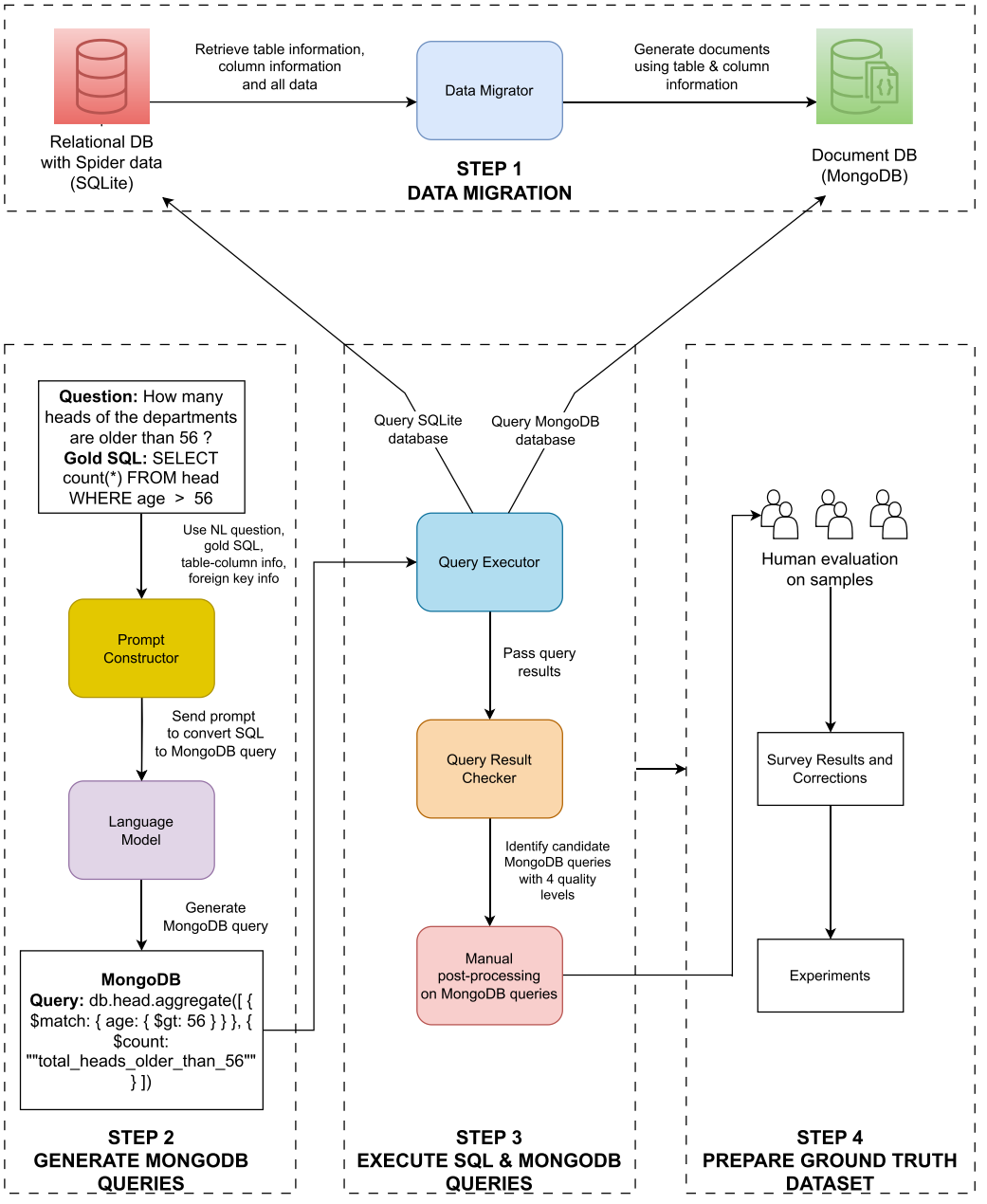

Figure 3. The proposed dataset construction and evaluation pipeline.

3.1 Data migration

The first problem that we addressed is migrating relational data from the relational database to the MongoDB database. Since the original Spider dataset was shared as SQLite databases,[f] we used them as the source to create databases and collections in MongoDB database server.

[f] https://www.sqlite.org/

Unlike SQLite tables, the MongoDB collections do not require data type definitions for columns. This enabled a migration process without any issue of type-compatibility.

During the migration of all SQLite databases, first all table names are found. Then all the table content is exported with column name information. Later table name is used as the collection name, and column names are used as field names in MongoDB database. Below, the SQL queries that were executed sequentially for the data migration are presented.

`SELECT name FROM sqlite_master WHERE type='table'`	retrieves the names of all tables in the SQLite database
`SELECT * FROM <table_name>`	retrieves all rows in the given table
`PRAGMA table_info(<table_name>)`	retrieves column names and column types from the given table

Note that, after exporting the relational data, we used the *PyMongo*[g] Python library to import the data into the MongoDB database. The migration step was needed to prepare the environment to run the generated MQL queries.

3.2 Generating SQL-equivalent MongoDB queries

We used LLMs that perform well on coding and math tasks to transform gold SQL queries to their equivalent MQL counterparts. To be more specific, instruction prompts were provided to the LLMs (GPT4 (Achiam *et al.* 2023), GPT3.5 (aka ChatGPT) (Brown *et al.* 2020), and DeepSeek Coder 33B (Guo *et al.* 2024)) for generating MQL queries. Instruction prompting is a technique that includes task-specific instructions to the model to explain the intention (Ye *et al.* 2023).

We provided prompts for each input in the Spider dataset, ensuring that they encompass schema information with foreign key relationships and questions. Additionally, the gold SQL statement is provided to guide the models, particularly for equality constraints, due to the case-sensitive nature of the data stored in databases. For example, when querying all female students, the model needs to understand how the gender field is formatted ("F", "Female," etc.). Since this information is only available in the gold SQL expression, we decided to include the gold SQL in the prompt. The template prompt format is presented in Figure 4a, and a sample prompt for MQL query generation is shown in Figure 4b. More example prompts for MQL generation via LLMs could be found at Appendix A.

3.3 Evaluation of the generated MongoDB queries

In the Spider challenge,[h] the performance of each model is evaluated on two metrics: *Exact Set Matching Accuracy* and *Execution Accuracy*. While exact set matching accuracy decomposes each SQL statement into several clauses and conducts set comparison in each SQL clause, execution accuracy focuses on if the predicted query result rows are identical to the actual query result rows. In this study, we evaluate the correctness of the generated MQL queries according to the **execution accuracy** metric.

At this stage, we executed the generated MQL queries on MongoDB and gold SQLs on SQLite database. It is noteworthy that the Spider's test set did not contain the gold SQLs. Therefore, MQL queries were generated for 7,000 training and 1,034 validation instances, for 166 different databases. To establish a match, the number of resulting rows must be identical in both gold and generated queries.

[g] https://pymongo.readthedocs.io/en/stable/
[h] https://yale-lily.github.io/spider

(a)

```
Write only the MongoDB with no explanation
for the query using the following
schema. Do not select extra columns
that are not explicitly requested.
Schema:
<table_1(column_1, column_2, ...)
<table_2(column_1 column_2, ...)
...
...
Foreign keys:
<table_1>.<column_1> = <table_2>.<column_2>
...
...
Question:
<NL question>
Gold  SQL:
<gold SQL query provided from Spider>
```

Prompt template

(b)

```
Write only the MongoDB with no explanation
for the query using the following
schema. Do not select extra columns
that are not explicitly requested.
Schema:
publication(Publication_ID, Book_ID,
Publisher, Publication_Date, Price)
book(Book_ID, Title, Issues, Writer)
Foreign keys:
publication.Book_ID = book.Book_ID
Question:
What are the dates of publications in
descending order of price?
Gold SQL:
SELECT Publication_Date
FROM publication
ORDER BY Price DESC
```

Example prompt

Figure 4. Prompt engineering for MQL query generation.

Execution accuracy requires a one-to-one match, and only MQL queries that return the same rows are considered at the beginning. However, when LLMs are used for MQL generation, it is seen that the generated queries sometimes return the same rows in different order, or the same rows with extra columns. To increase the number of MongoDB queries in the DocSpider ground-truth dataset, such queries are manually updated for correcting the ordering and projection and are included in the ground-truth dataset.

We defined four quality levels to classify generated MQL queries based on the query result equivalence:

- **Same:** Both query results should be identical. If the query results are in different order, but gold SQL does not have *order by* clause; these are also considered as identical. This is the most strict comparison metric among the four quality levels.
- **Extra fields:** The generated MQL query returns the same rows with additional fields.
- **Unordered:** Both query results must be the same, except they are ordered differently. In other words, the gold SQL statement includes the *order by* clause and the order of the result set of the generated MQL and gold SQL are different.
- **Unordered extra fields:** The MongoDB query returned extra fields and the query results have different ordering. This quality level is the lowest one acceptable for queries that can be included in the ground-truth dataset.

More specifically, if a query is classified as *extra_fields* (or *unordered_extra_fields*), it means there is a projection error to fix. If the MQL query result ordering is different than the SQLite query result ordering and the gold SQL query includes *order by* clause, then it means that there is an ordering error to fix, and the generated MQL is classified as *unordered* (or *unordered_extra_fields*). Subsequently, the queries were manually updated by adding/modifying:

- *$project* clauses for the queries satisfying *extra_fields* and *unordered_extra_fields* quality levels.
- *$sort* clauses for the queries satisfying *unordered* and *unordered_extra_fields* quality levels.

Table 1. Number of ground-truth queries generated by language models

Dataset	Model	Number of queries
Train	GPT4	3,670
Train	DeepSeek Coder 33B	2,818
Train-total	-	4,043
Dev	GPT4	554
Dev	DeepSeek Coder 33B	434
Dev	GPT3.5	371
Dev-total	-	620

This process increased the number of queries that can be used as a train set of DocSpider dataset. However, pure execution accuracy (one-to-one match, *same* quality level) is used for evaluation in the experiments conducted in Section 4.

After performing the corrections to the MQL queries generated by LLMs, the union of the generated MQL queries by different LLMs was collected. In Table 1, the number of successfully translated gold SQLs by each LLM are presented. For instance, GPT4 generated 3,670 SQL-equivalent queries, while DeepSeek Coder generated 2,818 SQL-equivalent queries. By taking the union of these, we obtained 4,043 SQL-equivalent MQL queries that can be considered as the gold MQL queries for the DocSpider dataset.

The Spider dataset classified the SQL queries into four difficulties: *easy*, *medium*, *hard,* and *extra hard*. The hardness of each query was determined according to the number of SQL components and query structure (nested structure, keywords, and aggregations). The difficulty level distribution of the original Spider dataset and DocSpider dataset are presented in Figure 5. Even though we cannot fully transform the Spider dataset into MQL queries using LLMs, the difficulty level distribution of the DocSpider dataset is very similar to that of the Spider dataset.

3.4 User study

We considered the generated MQL queries as "gold MQLs" if they satisfied any of the given quality levels and afterward had the final manual correction. However, to assess the correctness of these queries, the generated MQL queries need to be evaluated by human reviewers. To this aim, 300 sample queries out of 4,043 were selected from the corrected set, and we aim to assess the truthfulness of the generated MQL queries considering the prompt given in Section 3.2. The annotators marked incorrect queries as *false* and the correct queries as *true*. The queries were selected randomly, but the hardness distribution was kept similar to the DocSpider dataset distribution.

The annotators were selected from graduate students who had a Computer Science background and had passed the Database Management course. We had six annotators and each annotator was given 110 queries to review where 100 queries were from DocSpider and 10 queries from incorrect MQL queries (generated by LLMs). The annotators were not informed about which of the given queries were incorrect. In addition, each query was annotated by at least two reviewers such that if there was a conflict between a pair of reviewers on a specific instance, then we asked a third meta-reviewer to review the instance and applied the majority voting. As a result of the user study, only 6 out of 300 queries from the DocSpider dataset were marked as incorrect.

Note that six queries annotated as incorrect returned the same query results with the gold SQL queries due to the sparsity of sample instances in the Spider databases. For instance,

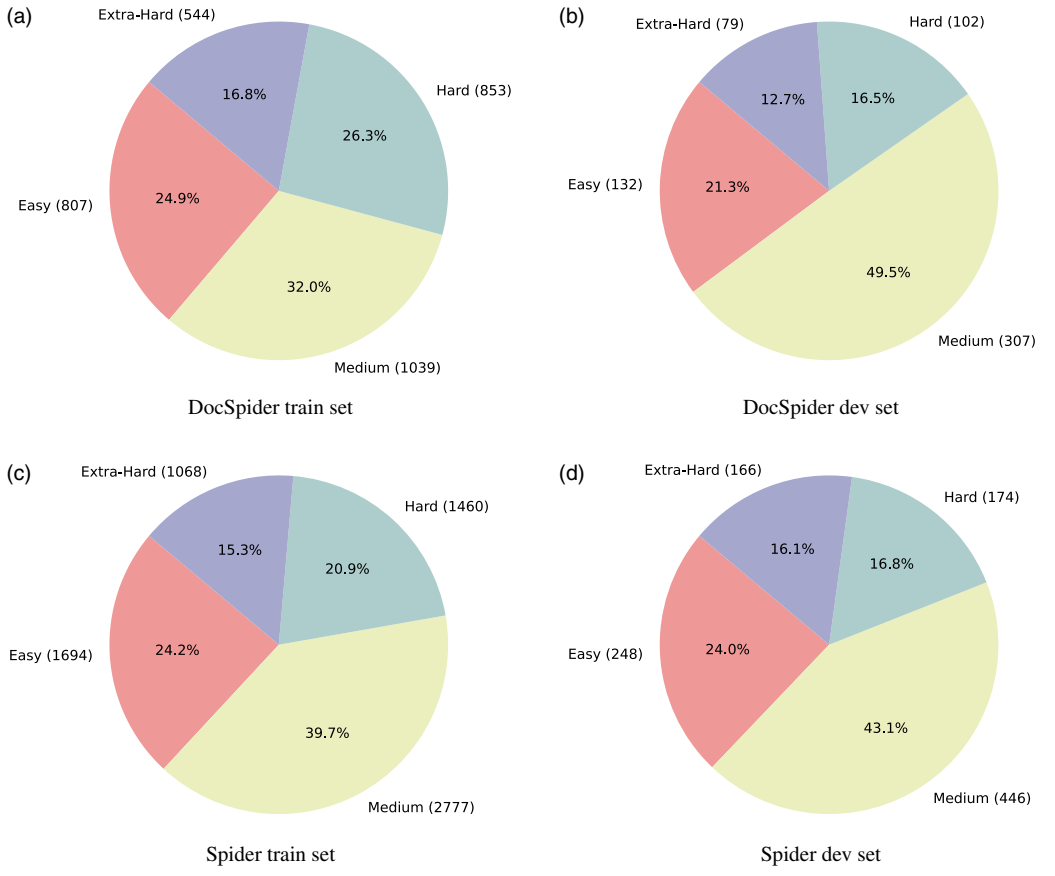

Figure 5. Difficulty level distribution of the queries in the data splits.

the following gold SQL query in the Spider dataset was translated into the MongoDB query provided below:

```
SELECT DISTINCT product_details
FROM products
```

```
db. Products.find({},
{product_details: 1, _id: 0})
```

Even though the generated MQL query retrieves *product_details* from the *products* collection, it does not ensure that the returned values in the *product_details* field are distinct. It fetches all documents without eliminating duplicates.

Furthermore, it has been observed that the Spider dataset, despite being manually curated, contains some ambiguous instances. Consider the following sample:

```
List the name, date and result
of each battle.
```

```
SELECT name, date FROM battle
```

In this instance, although the name, date, and result of each battle are requested, the gold SQL query does not include the result of the battles. Therefore, the DocSpider dataset might include some confusing instances. According to the user study, approximately 2% of the ground-truth MQL queries may be incorrect. This ambiguity can be addressed using the Reciprocal Rank Fusion

RAG approach Rackauckas (2024) which generates multiple queries for a single input and re-ranks them based on their reciprocal scores. Another promising method is Step-Back Prompting Zheng *et al.* (2024b) which involves generating a more abstract version of the user query to retrieve more relevant information for generating a response. However, this phenomenon needs a more in-depth exploration which we address as future work.

Finally, we assessed inter-annotator agreement to ensure the consistency of annotations. Cohen's Kappa (Cohen 1960) serves as the primary pairwise measure for inter-annotator agreement. The Cohen's κ scores for each question set were computed as 0.45, 0.64, and 0.81, indicating strong agreement in the second and third question sets, and reasonable agreement in the first set.

4. Experiments

In this section, we present the conducted experiments to assess the potential usefulness of this dataset for tackling the text-to-NoSQL problem. The following experiments were conducted using the DocSpider dataset:

1. Assessing LLM execution accuracy with zero-shot learning. The objective is to determine whether LLMs are already capable of translating NL queries into MQL queries or a text-to-NoSQL dataset is necessary for solving the text-to-NoSQL problem.
2. Text-to-SQL-to-NoSQL: Whether transforming text-to-NoSQL problem to SQL-to-NoSQL problem makes an improvement on execution accuracy scores. This is for understanding how successful LLMs would be, when SQL outputs from text-to-SQL model is used to reinforce LLMs on text-to-NoSQL task.
3. Assessing LLM execution accuracy by fine-tuning with DocSpider dataset. The objective is to verify that the DocSpider dataset is useful and provides significant improvement on execution accuracy scores for solving NL query to MQL query problem.
4. Verifying the usefulness of the DocSpider dataset by testing for a collection with nested-structured entries. The objective for this experiment is to analyze the effectiveness of the models fine-tuned with DocSpider dataset for nested MQL queries.

The following subsections explain the environment setup for the experiments. Also we share the experiment results as tables in these subsections. The detailed discussion about experiment results and our prominent observations about the performance of the employed LLMs can be found in Section 5.

4.1 Setup

In the experiments, Mistral 7B (Jiang *et al.* 2023), DeepSeek Coder 33B (Guo *et al.* 2024), GPT3.5 (gpt-3.5-turbo-0613), GPT4o (gpt-4o-2024-08-06), and CodeLlama 70B (Roziere *et al.* 2024) language models were employed. While selecting language models, we took into consideration their coding scores on the MT-Bench (Zheng *et al.* 2023) and Spider challenge leaderboard. Moreover, we intentionally selected both open-source LLMs such as DeepSeek Coder 33B and CodeLlama and proprietary LLM such as GPT3.5 and GPT4o to discuss the pros and cons of using open-source LLM for this task. Additionally, the Mistral-7B was chosen to assess the performance of a small, multi-purpose language model for text-to-NoSQL. The models were downloaded from the Huggingface's[i] repository and were evaluated on the execution accuracy metric, and the results after hyperparameter tuning are reported. The details about hyperparameter tuning can be found in Section 4.4.

[i] https://huggingface.co/

Table 2. Zero-shot execution accuracy percentages

Model	Overall exec.accuracy	Easy	Medium	Hard	Extra hard
Mistral 7B	20.4%	26.5%	26.4%	9.8%	0%
DeepSeek Coder 33B	29.2%	27.3%	35.8%	20.6%	17.7%
CodeLlama 70B	42.0%	43.9%	46.3%	38.2%	26.6%
GPT3.5	45.4%	45.5%	53.1%	39.2%	22.8%
GPT4o	**71.0%**	**76.5%**	**73.3%**	**66.6%**	**45.5%**

```
Write only the MongoDB with no explanation for the query
using the following schema. Do not select extra
columns that are not explicitly requested.
Schema:
<table_1(column_1, column_2, ...)
<table_2(column_1 column_2, ...)
Foreign keys:
<table_1>.<column_1> = <table_2>.<column_2>
...
Question:
<NL question>
```

Figure 6. Prompt template used in the experiments.

In the experiments, top-p sampling (Holtzman *et al.* 2020) decoding strategy was employed while generating tokens. Sampling decoding generates the next token according to its conditional probability. In top-p sampling, the smallest set of words whose cumulative distribution is above the threshold probability is considered while generating tokens. The number of words dynamically changes according to the next token's probability distribution. We set the probability threshold at 0.9 in the experiments.

The generic prompt template provided to the LLMs in the experiments is given in Figure 6. In addition, more examples can be seen at Appendix B. Recalling from Section 3.2, while preparing DocSpider dataset, we provided the gold SQL expression in the prompt. However, for the text-to-NoSQL experiments, we removed the gold SQL expression from the prompt, to experiment the real-life scenario for NLQ problem.

In the subsequent experiments, we utilized DocSpider's development set to evaluate the language models. Similarly, during fine-tuning, the training set was randomly split into 90% for actual training and 10% for validation. Furthermore, the exact execution accuracy values (precision level="same") were calculated in the following experiments.

4.2 Experiment 1: zero-shot text-to-NoSQL results

The goal of this experiment is to evaluate the baseline performance of certain LLMs on text-to-NoSQL task. More specifically, we provided the prompt template given in Section 4.1 to the models and expecting them to generate the corresponding MQL queries. This experiment is performed against Spider's dev dataset. As shown in Table 2, GPT4o outperformed the open-source LLMs on execution accuracy by a significant margin. GPT-4o has approximately 1 trillion parameters which enable it to learn more complex information, and this leads to an enhanced performance across various tasks.

Table 3. Text-to-SQL-to-NoSQL execution accuracy percentages

Model	Overall exec.accuracy	Easy	Medium	Hard	Extra hard
DAIL-SQL + Rule-based converter	26.8%	42.4%	30.0%	14.7%	3.7%
DAIL-SQL + Mistral 7B	29.4%	29.5%	42.0%	12.7%	1.3%
DAIL-SQL + DeepSeek Coder 33B	45.8%	50.0%	51.8%	36.3%	27.8%
DAIL-SQL + CodeLlama 70B	49.4%	44.7%	57.7%	46.1%	29.1%
DAIL-SQL + GPT3.5	**58.6%**	**59.1%**	**65.4%**	**55.9%**	**35.4%**

4.3 Experiment 2: text-to-SQL-to-NoSQL results

In this experiment, we explored the potential improvement by leveraging existing text-to-SQL solutions. Therefore, we formulated the text-to-NoSQL problem as a text-to-SQL-to-NoSQL problem.

As the name implies, text-to-SQL-to-NoSQL pipeline consists of two steps. In the first step, we employed the DAIL-SQL (Gao *et al.* 2024) model, which achieved 86.2% execution accuracy on Spider's unseen test set, to translate the text inputs to SQL expressions. DAIL-SQL model gives the highest accuracy so far[j] in the Spider challenge[k] at the time of writing this paper.

As the second step, the generated SQL expressions are translated to MQL statements. To this aim, we used a rule-based conversion and the also employed a set of LLMs. We searched for a comprehensive model that converts SQL queries to MQL queries, but only a couple of rule-based web tools (Russell 2016; Kotzen 2020) were available. We employed Russell (2016)'s converter model to evaluate its performance against language models. Note that the SQL queries generated by DAIL-SQL were provided as input to the converter.

Furthermore, we utilized LLMs to carry out SQL-to-NoSQL task. Similarly, DAIL-SQL's predicted queries were provided to LLMs to generate MQL queries. To fulfill this, we replaced the "Question" field in the prompt (given in Section 4.1) with the generated SQL expression and asked the language models to transform it into an equivalent MQL statement. According to Table 3, GPT3.5 demonstrated superior performance compared to open-source models across all hardness levels. In addition, the models that were specifically pretrained for coding tasks performed better than Mistral 7B on SQL-to-NoSQL generation. This outcome was expected, given that the tokenizer in Mistral-like, multi-purpose language models may struggle to tokenize SQL clauses accurately.

4.4 Experiment 3: fine-tuning results

We used three Nvidia A100 GPUs with 40 GB VRAM, setting the maximum sequence length to 4096 tokens, a batch size of 8 for Mistral and 4 for DeepSeek-Coder and trained for 3 epochs. The train/validation loss graphs are presented in Appendix D. Note that we used the checkpoints with the minimum validation loss for the inference.

First of all, we fine-tuned GPT3.5 (gpt-3.5-turbo-0613) and GPT4o (gpt-4o-2024-08-06) for 3 epochs on Open AI's API. For the open-source models, we used the Huggingface's Trainer class which allows instruction format and conversational fine-tuning. Besides it supports parameter-efficient fine-tuning with QLoRA (Dettmers *et al.* 2023). The batch size was set to 4, epochs to 3, the initial learning rate to 1e-4 with linear weight scheduler, and the Adam optimizer was used.

[j] Highest accuracy among the studies with their source code shared.
[k] https://yale-lily.github.io/spider

Table 4. Fine-tuning execution accuracy percentages

Model	Overall exec.accuracy	Easy	Medium	Hard	Extra hard
Mistral 7B	40.7%	49.2%	41.4%	28.4%	39.2%
DeepSeek Coder 33B	70.8%	84.1%	73.9%	65.7%	41.8%
GPT3.5	71.1%	78.8%	70.4%	70.6%	60.8%
GPT4o	**81.1%**	**91.6%**	**82.1%**	**77.5%**	**64.5%**

Table 5. The average cosine similarity values for correct and incorrect MQLs for fine-tuned models

Model	Correctly predicted MQLs	Incorrectly predicted MQLs
Mistral 7B	0.976	0.923
DeepSeek Coder 33B	0.971	0.930
GPT3.5	0.974	0.908
GPT4o	0.964	0.941

While fine-tuning the language models (in this experiment), we employed the QLoRA technique which efficiently quantizes the frozen model weights to 4-bit NormalFloat and integrates LoRA (Hu *et al.* 2021) adapters to the model. LoRA is a parameter-efficient fine-tuning technique that decomposes a model's weight matrix into smaller rank matrices. In this way, LoRA introduces a small set of new parameters and these parameters are only fine-tuned, which provides more compute and resource-efficient training of LLMs without a significant performance loss. We set the dimension of the low-rank matrices (r) to 32, establishing the scaling factor for the weight matrices (lora_alpha) at 32, and specifying a dropout probability of 0.05 for the LoRA layers (lora_dropout).

In Table 4, execution accuracy results after fine-tuning with the DocSpider dataset are presented. According to the results, the language models that were reported to perform well on coding tasks such as GPT3.5, GPT4o, and DeepSeek Coder 33B performed above 70%. Additionally, DeepSeek Coder, which is an open-source language model, performed better than GPT3.5 on easy and medium instances. Last but not least, even though Mistral 7B is not specifically pretrained for coding tasks, it achieved 40.7% execution accuracy after fine-tuning with a considerably small text-to-NoSQL dataset.

Text embedding-based evaluation

In addition to using execution accuracy metric for text-to-NoSQL task, we performed an analysis with the cosine similarity metric in order to evaluate the LLM performance. To achieve this, predicted MQLs are generated by using the DocSpider test set. First, using Reimers (2019)'s SBERT model, predicted and actual MQLs are considered as sentences and embedding vectors are obtained for MQLs. Then, for each predicted-actual MQL embedding vector pair, the cosine similarity is calculated. Following this, the average of the cosine similarity values for the correct and incorrect MQLs are calculated separately. The results show that the that correctly predicted MQLs have higher average cosine similarity value than the incorrectly predicted MQLs, for fine-tuned models. The fine-tuned models and their average cosine similarity values are presented in 5.

4.5 Experiment 4: nested queries

In relational databases, it is a common practice to a express a relationship between two databases through primary key—foreign key pairs, instead of storing the table in the joined form. This is found to be preferable due to normalization considerations.

Unlike relational databases, MongoDB collections are capable of storing nested documents. Besides a single value, a field can hold an object, which can be considered as a reference to another table in a relational database. This leads to nested structures and nested queries on such collections.

In order to analyze the effectiveness of the DocSpider dataset to fine-tune an LLM to handle nested structures and nested query translation, another experiment is conducted. In this experiment, GPT3.5 fine-tuned with DocSpider dataset is used for the text-to-MQL task. This model is chosen since it gives the highest performance improvement (from 45.4% to 71.1%) in fine-tuning analysis on DocSpider test collections (as given in Table 2 and Table 4). As a native database having a nested structure, the sample dataset shared in MongoDB's website[1] is used for the evaluation. Twenty queries are hand-crafted for this experiment, details of which are given in Appendix C.

The query generation performance of GPT3.5 fine-tuned with DocSpider dataset is compared against base GPT3.5 model. It is seen that the base GPT3.5 model is able to generate nine queries correctly out of twenty test queries, whereas the fine-tuned model generated thirteen of them successfully out of twenty. Therefore, the query generation accuracy of the base model is still about 45%, whereas with the fine-tuning using DocSpider, the accuracy increases to 65% for nested structures. Hence, it is seen that the DocSpider dataset is effective for fine-tuning an LLM to improve performance on generating nested MongoDB queries.

5. Discussion

The primary goal of this study has been to explore the need for a cross-domain text-to-NoSQL dataset for MongoDB. Additionally, we have evaluated the impact of pretrained knowledge bases in LLMs on addressing this issue. While our emphasis has been on MongoDB in this study, the methodology can be adapted to similar tasks and databases. However, the success of the models may differ based on the pretrained knowledge embedded in LLMs and the syntactic similarity of query languages to SQL-like structures.

While looking at the results of the Experiment 1 given in Table 2, it is hard to say that base LLMs are successful on text-to-NoSQL task. Due to model sizes, CodeLLama 7B (42%), GPT3.5 (45.4%), and GPT4o (71%) had reasonably higher execution accuracy than Mistral 7B (20.4%) and DeepSeek Coder 33B (29.2%). The best model for zero-shot learning on text-to-NoSQL task appears as the base GPT4o model with 71% execution accuracy; however, its overall execution accuracy is far from being successful from the NLQ problem perspective. Hence, the base LLMs capabilities are shown to be limited for the text-to-NoSQL task.

Another prominent observation is that, as given in the Experiment 2 highlighted in Table 3, in the absence of an annotated NoSQL dataset, reframing the problem as text-to-SQL-to-NoSQL consistently yields superior results compared to the zero-shot text-to-NoSQL approach. In other words, if the outputs of a text-to-SQL model is fed to a model with SQL-to-NoSQL translation capabilities, then whole system is more accurate than zero-shot learning with base LLMs on text-to-NoSQL translation task. When the results of the Experiment 2 are inspected compared to the Experiment 1 results, a consistent increase is observed for each of the models: Mistral 7B's accuracy from 20.4% to 29.4%; DeepSeek Coder 33B's from 29.2% to 45.8%; CodeLlama 7B's from 42% to 49.4% and lastly GPT3.5's accuracy from 45.4% to 58.6%. While GPT3.5 has the

[1]https://www.mongodb.com/docs/atlas/sample-data/sample-airbnb/

best results for the Experiment 2, DeepSeek Coder 33B's translation performance is also vastly improved by $\sim$ 56%.

Furthermore, the results of the Experiment 3 in Table 4 clearly shows the positive effect of having a text-to-NoSQL dataset to fine-tune an LLM in order to solve the NLQ problem for NoSQL databases. When the results are compared against the Experiment 2 in Table 3, it is observed that although constructing a text-to-SQL-to-NoSQL solution could give moderate results, far better results are achievable by fine-tuning with a dataset specialized for the text-to-NoSQL task. A fine-tuned LLM on text-to-NoSQL data can outperform any type of text-to-SQL-to-NoSQL solution, whether it is a rule-based converter, or a text-to-SQL solution supported with an LLM.

Comparing the results of the Experiment 3 against the Experiment 1 in Table 2, it is seen that Mistral 7B's accuracy increased from 20.4% to 40.7%, DeepSeek Coder 33B's from 29.2% to 70.8%, GPT3.5's accuracy increased from 45.4% to 71.1%, and lastly GPT4o's accuracy increased from 71% to 81.1%. Fine-tuning by the DocSpider dataset provided performance improvements by $\sim$ 100% on Mistral 7B, by $\sim$ 140% on DeepSeek Coder 33B, by $\sim$ 36% on GPT-3.5, and by $\sim$ 14% on GPT-4o. Another important observation is that, despite the DocSpider dataset's modest size, through fine-tuning (40.7%), Mistral 7B competes with the zero-shot performance of the ten times larger CodeLlama 7B (42.0%).

In addition, in all experiments, the LLMs that were either specifically pretrained on coding tasks (CodeLlama 7B and DeepSeek Coder 33B) or general-purpose LLMs that performed well on coding tasks (GPT3.5 and GPT4o) consistently outperformed Mistral 7B. It can be attributed to their pretrained knowledge, that is, during pretraining they may have learned the MongoDB query structure from examples. In contrast, Mistral and similar LLMs might be pretrained more on conversational and dialog generation tasks. Consequently, when selecting a language model, it becomes crucial to prefer a model whose pretraining objective aligns with the intended task.

Finally, we assessed the proposed dataset using both open-source and proprietary LLMs. Given the resource-intensive nature of CodeLlama and GPT, fine-tuned versions of Mistral 7B or DeepSeek Coder 33B could serve as viable alternatives (specifically DeepSeek Coder 33B, with 70.8% accuracy). Fine-tuning a model such as GPT is a resource-intensive process—potentially hitting financial, computational, and hardware constraints—and an open-source language model can be utilized to achieve competitive results.

6. Conclusion

Interacting with databases by using NL has been a challenge for a long time. Several studies addressed this problem by converting NL queries into SQL statements on relational databases. Utilizing cross-domain datasets such as Spider for text-to-SQL task, and with the advancements in LLMs, major improvements have been achieved on the text-to-SQL problem. However, for the text-to-NoSQL task, the obtained improvements are still limited.

In this study, we introduce the DocSpider dataset, the first cross-domain text-to-NoSQL dataset for the MongoDB document database. MongoDB is selected as the target database considering its increasing use as the data store in software projects. During the MongoDB query generation phase, we performed instruction prompting by including NL query, table-column information, foreign keys, and gold SQL statements to utilize LLMs. By applying manual post-processing on the generated MongoDB queries, we obtained 4,043 NL–MongoDB query pairs from Spider train dataset, and 620 NL–MongoDB query pairs from Spider development dataset and included these question–query pairs into DocSpider dataset.

Our approach for text-to-NoSQL query generation from a text-to-SQL dataset is adaptable and can be extended to other NoSQL databases. Subsequently, we demonstrated the necessity of a text-to-NoSQL dataset through comparisons with cross-language transfer learning and in-context learning approaches. Our experiments demonstrated that fine-tuning LLMs with the

DocSpider dataset significantly improved the query translation accuracy compared to their base versions. To be more specific, Mistral 7B's accuracy increased from 20.4% to 40.7%, DeepSeek Coder 33B's accuracy rose from 29.2% to 70.8%, GPT-3.5's accuracy improved from 45.4% to 71.1%, and GPT-4's accuracy increased from 71.0% to 81.1%. Additionally, DocSpider's usefulness is also confirmed for querying nested-structured MongoDB collections. The related experiment demonstrated a performance improvement from 45% to 65% when utilizing the GPT-3.5 model.

Reliance on language models susceptible to hallucination (Zhang *et al.* 2023b) may certainly be a limitation for our approach. Context-aware decoding can be a promising way to mitigate this problem (Shi *et al.* 2023; Tonmoy *et al.* 2024). For query translation evaluation, we used the execution accuracy metric, which compares the outputs of two given queries. This provides a more practical means of verification than the sub-query comparisons. However, there could be alternative answers that yield the same results with an optimal query structure, involving fewer operations.

Last but not least, we have not observed major error patterns in generated MQL statements. However, errors tend to be more frequent with less common syntactic structures, such as queries requiring NOT IN or those containing regular expressions. We plan to address this issue further in future work to determine if some of these errors can be corrected through postprocessing.

Acknowledgments. This research received the support of the EXA4MIND project, funded by the European Union's Horizon Europe Research and Innovation Programme, under Grant Agreement N° 101092944. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them. The work is also supported by EuroHPC Development Access Call with Project ID DD-23-122. We would like to thank the research assistants Aslı Umay Öztürk, Can Ünaldı, and Yiğit Sever from the Department of Computer Engineering in METU for their suggestions and contributions to the user study.

References

Abdelhedi F., Brahim A. A. and Zurfluh G. (2021). Ocl constraints checking on nosql systems through an mda-based approach. *International Journal of Data Warehousing and Mining (IJDWM)* **17**(1), 1–14.

Achiam J., Adler S., Agarwal S., Ahmad L., Akkaya I., Leoni Aleman F., Almeida D., Altenschmidt J., Altman S., Anadkat S., Avila R., Babuschkin I., Balaji S., Balcom V., Baltescu P., Bao H., Bavarian M., Belgum J., Bello I., ... Radford A. (2023). GPT-4 Technical Report. arXiv preprint arXiv:2303.08774

Akhtar A. (2023). Popularity ranking of database management systems. arXiv preprint arXiv: 2301.00847

Bay Y. Y. and Yearick K. A. (2024). Machine learning vs deep learning: the generalization problem. arXiv preprint arXiv: 2403.01621

Bazaga A., Gunwant N. and Micklem G. (2021). Translating synthetic natural language to database queries with a polyglot deep learning framework. *Scientific Reports* **11**(1), 18462.

Bhagavatula C. S., Noraset T. and Downey D. (2015). Tabel: Entity linking in web tables. In *International Semantic Web Conference*, Cham: Springer International Publishing, pp. 425–441.

Blank S., Wilhelm F., Zorn H.-P. and Rettinger A. (2019). Querying nosql with deep learning to answer natural language questions. *Proceedings of the AAAI Conference on Artificial Intelligence* **33**(1), 9416–9421.

Brown T. B., Mann B., Ryder N., Subbiah M., Kaplan J., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A., Agarwal S., Herbert-Voss A., Krueger G., Henighan T., Child R., Ramesh A., Ziegler D. M., Wu J., Winter C., Hesse C., Chen M., Sigler E., Litwin M., Gray S., Chess B., Clark J., Berner C., McCandlish S., Radford A., Sutskever I. and Amodei D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, vol. 2, Curran Associates, Inc, pp. 1877–1901.

Chen Z., Gu Z., Cao L., Fan J., Madden S. and Tang N. (2023). Symphony: Towards natural language query answering over multi-modal data lakes, In *13th Conference on Innovative Data Systems Research, CIDR 2023*, Amsterdam, The Netherlands, pp. 8–151. www.cidrdb.org.

Cohen J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement* **20**(1), 37–46.

Dechter R. and Pearl J. (1985). Generalized best-first search strategies and the optimality of a. *Journal of the ACM (JACM)* **32**(3), 505–536.

Dettmers T., Pagnoni A., Holtzman A. and Zettlemoyer L. (2023). Qlora: efficient finetuning of quantized LLMs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY: Curran Associates Inc.

Devlin J., **Chang M.-W.**, **Lee K. and Toutanova K.** (2019). BERT: pre-training of deep bidirectional transformers for-language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*, June 2-7, 2019, Vol. 1 (Long and Short Papers), Minneapolis, MN: Association for Computational Linguistics, pp. 4171–4186.

Dong X., **Zhang C.**, **Ge Y.**, **Mao Y.**, **Gao Y.**, **Lin J.**, **Lou D.** (2023). C3: zero-shot text-to-sql with chatgpt. arXiv preprint arXiv: 2307.07306

Gao D., **Wang H.**, **Li Y.**, **Sun X.**, **Qian Y.**, **Ding B. and Zhou J.** (2024). Text-to-sql empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment* **17**(5), 1132–1145.

Guo D., **Zhu Q.**, **Yang D.**, **Xie Z.**, **Dong K.**, **Zhang W.**, **Chen G.**, **Bi X.**, **Wu Y.**, **Li Y. K.**, **Luo F.**, **Xiong Y. and Liang W.** (2024). Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *CoRR*, abs/2401.14196.

Hains G. J., **Khmelevsky Y. and Tachon T.** (2019). From natural language to graph queries, In *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*. IEEE, pp. 1–4

Holtzman A., **Buys J.**, **Du L.**, **Forbes M. and Choi Y.** (2020). The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Hossen K. M., **Uddin M. N.**, **Arefin M. and Uddin M. A.** (2023). BERT model-based natural language to nosql query conversion using deep learning approach. *International Journal of Advanced Computer Science and Applications* **14**(2)

Hu E. J., **Shen Y.**, **Wallis P.**, **Allen-Zhu Z.**, **Li Y.**, **Wang S.**, **Wang L. and Chen W.** (2021). Lora: low-rank adaptation of large language models. *CoRR*, abs/2106.09685.

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

Jung H. and Kim W. (2020). Automated conversion from natural language query to sparql query. *Journal of Intelligent Information Systems* **55**(3), 501–520.

Kang Y.-S., **Park I.-H.**, **Rhee J. and Lee Y.-H.** (2015). MongoDB-based repository design for IoT-generated RFID/sensor big data. *IEEE Sensors Journal* **16**(2), 485–497.

Kate A., **Kamble S.**, **Bodkhe A. and Joshi M.** (2018). Conversion of natural language query to SQL query, In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE, pp. 488–491

Kobeissi M., **Assy N.**, **Gaaloul W.**, **Defude B.**, **Benatallah B. and Haidar B.** (2023). Natural language querying of process execution data. *Information Systems* **116**, 102227.

Kotzen R. (2020). SQL to MongoDB query converter. https://github.com/synatic/noql/. (accessed 2024).

Lewis M., **Liu Y.**, **Goyal N.**, **Ghazvininejad M.**, **Mohamed A.**, **Levy O.**, **Stoyanov V. and Zettlemoyer L.** (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, pp. 7871–7880.

Li J., **Hui B.**, **Qu G.**, **Yang J.**, **Li B.**, **Li B.**, **Wang B.**, **Qin B.**, **Geng R.**, **Huo N.**, **Zhou X.**, **Ma, C.**, **Li G.**, **Chang K. C.**, **Huang F.**, **Cheng R. and Li Y.** (2024). Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*, Red Hook, NY: Curran Associates Inc.

Li Y., **Yang H. and Jagadish H. V.** (2005). Nalix: an interactive natural language interface for querying xml, In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, New York, NY: Association for Computing Machinery, pp. 900–902.

Mahmood K. and Risch T. (2021). Scalable real-time analytics for iot applications, In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, Los Alamitos, CA: IEEE, IEEE Computer Society, pp. 404–406.

Majeed M. T., **Ahmad M. and Khalid M.** (2016). Automated XQuery generation for nosql. In *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*. IEEE, pp. 507–512

Makris A., **Tserpes K.**, **Spiliopoulos G.**, **Zissis D. and Anagnostopoulos D.** (2021). MongoDB vs PostgreSQL: a comparative study on performance aspects. *GeoInformatica* **25**, 243–268.

Membrey P., **Plugge E.**, **Hawkins D. and Hawkins T.** (2010). *The Definitive Guide to MongoDB: the noSQL Database for Cloud and Desktop Computing*. Springer

Mondal S., **Mukherjee P.**, **Chakraborty B. and Bashar R.** (2019). Natural language query to nosql generation using query-response model. In *2019 International Conference on Machine Learning and Data Engineering (iCMLDE)*, Los Alamitos, CA: IEEE, IEEE Computer Society, pp. 85–90.

Owei V., **Rhee H.-S. and Navathe S.** (1997). Natural Language Query Filtration in the Conceptual Query Language. In *2014 47th Hawaii International Conference on System Sciences*, vol. 3, Los Alamitos, CA: IEEE Computer Society, pp. 539–549.

Paparizos S., **Ntoulas A.**, **Shafer J. and Agrawal R.** (2009). Answering web queries using structured data sources. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, New York, NY: Association for Computing Machinery, pp. 1127–1130.

Pennington J., **Socher R. and Manning C. D.** (2014). GloVe: Global vectors for word representation, In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543.

Peters M. E., Neumann M., Iyyer M., Gardner M., Clark C., Lee K. and Zettlemoyer L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1 (Long Papers), New Orleans, LA: Association for Computational Linguistics, pp. 2227–2237.

Pradeep T., Rafeeque P. and Murali R. (2019). Natural language to nosql query conversion using deep learning, In *International Conference on Systems, Energy & Environment (ICSEE)*, 2019, GCE Kannur, Kerala, July 2019.

Rackauckas Z. (2024). Rag-fusion: a new take on retrieval-augmented generation. *International Journal on Natural Language Computing (IJNLC)* **13**(1), 37–47.

Raffel C., Shazeer N., Roberts A., Lee K., Narang S., Matena M., Zhou Y., Li W. and Liu P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* **21**(1), 5485–5551.

Rai D., Wang B., Zhou Y. and Yao Z. (2023). Improving generalization in language model-based text-to-SQL semantic parsing: two simple semantic boundary-based techniques. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, vol. 2 (Short Papers), Toronto, Canada: Association for Computational Linguistics, pp. 150–160.

Reimers N. and Gurevych I. (2019). Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Conference on Empirical Methods in Natural Language Processing*

Rozière B., Gehring J., Gloeckle F., Sootla S., Gat I., Tan X. E., Adi Y., Liu J., Sauvestre R., Remez T., Rapin J., Kozhevnikov A., Evtimov I., Bitton J., Bhatt M., Ferrer C. C., Grattafiori A., Xiong W., Défossez A., Copet J., Azhar F., Touvron H., Martin L., Usunier N., Scialom T. and Synnaeve G. (2024). Code Llama: Open foundation models for code.

Russell V. (2016). SQL to MongoDB query converter. https://github.com/vincentrussell/sql-to-mongo-db-query-converter/ (accessed 2024).

Sak H., Senior A. W. and Beaufays F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, pp. 338–342.

Scholak T., Schucher N. and Bahdanau D. (2021). PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic, Association for Computational Linguistics, pp. 9895–9901.

Shi W., Han X., Lewis M., Tsvetkov Y., Zettlemoyer L. and tau Yih S. W. (2023). Trusting your evidence: hallucinate less with context-aware decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 2 (Short Papers), Mexico City, Mexico: Association for Computational Linguistic, pp. 783–791.

Tonmoy S. M. T. I., Zaman S. M. M., Jain V., Rani A., Rawte V., Chadha A. and Das A. (2024). A comprehensive survey of hallucination mitigation techniques in large language models.

Uma M., Sneha V., Sneha G., Bhuvana J. and Bharathi B. (2019). Formation of sql from natural language query using nlp. In 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), IEEE, pp. 1–5.

Xu X., Liu C. and Song D. (2017). SQLNet: generating structured queries from natural language without reinforcement learning. ArXiv, abs/1711.04436.

Ye S., Hwang H., Yang S., Yun H., Kim Y. and Seo M. (2023). Investigating the effectiveness of task-agnostic prefix prompt for instruction following. In *NeurIPS. 2023 Workshop On Instruction Tuning and Instruction Following*

Yin P., Lu Z., Li H. and Kao B. (2016). Neural enquirer: learning to query tables in natural language. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, pp. 2308–2314.

Yu T., Zhang R., Er H., Li S., Xue E., Pang B., Lin X. V., Tan Y. C., Shi T., Li Z. and et al. (2019). Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 1962–1979.

Yu T., Zhang R., Yang K., Yasunaga M., Wang D., Li Z., Ma J., Li I., Yao Q., Roman S. and et al. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 3911–3921.

Zhang W., Zeng K., Yang X., Shi T. and Wang P. (2023a). Text-to-esq: A two-stage controllable approach for efficient retrieval of vaccine adverse events from nosql database. In Proceedings of the 14th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 1–10.

Zhang Y., Li Y., Cui L., Cai D., Liu L., Fu T., Huang X., Zhao E., Zhang Y., Chen Y., Wang L., Luu A. T., Bi W., Shi F. and Shi S. (2023b). Siren's song in the ai ocean: a survey on hallucination in large language models. *ArXiv* abs/2309.01219.

Zhao Z., Liu W., French T. and Stewart M. (2023). Cyspider: A neural semantic parsing corpus with baseline models for property graphs. In Australasian Joint Conference on Artificial Intelligence, Springer, pp. 120–132.

Zheng D., Lapata M. and Pan J. Z. (2024a). Archer: A human-labeled text-to-SQL dataset with arithmetic, commonsense and hypothetical reasoning. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics*, vol. 1 (Long Papers), St. Julian's, Malta: Association for Computational Linguistics, pp. 94–111.

Zheng H. S., Mishra S., Chen X., Cheng H.-T., Chi E. H., Le Q. V. and Zhou D. (2024b). Take a step back: Evoking reasoning via abstraction in large language models.

Zheng L., Chiang W.-L., Sheng Y., Zhuang S., Wu Z., Zhuang Y., Lin Z., Li Z., Li D., Xing E. P., Zhang H., Gonzalez J. E. and Stoica I. (2023). Judging LLM-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*, Red Hook, NY: Curran Associates Inc.

Zhong V., Xiong C. and Socher R. (2017). Seq2SQL: generating structured queries from natural language using reinforcement learning.

Appendix

A. Prompt examples for MQL generation

Below are the sample queries in different difficulty levels: *"easy*," "*medium*," "*hard*," and "*extra hard."* These prompts are given to LLMs for the Spider train dataset. Recalling from Section 3.2, gold standart SQLs are given in the prompts to improve the MongoDB query generation. Later, a postprocessing is applied (depending on the precision level mentioned in Section 3.3) on the generated MQL query.

A.1 Query #1 (easy)

```
Write only the SQL with no explanation for the query using the following
schema. Do not select extra columns that are not explicitly requested.
<</SYS>>
Schema:
department(Department_ID, Name, Creation, Ranking, Budget_in_Billions,
  Num_Employees)
head(head_ID, name, born_state, age)
management(department_ID, head_ID, temporary_acting)
Foreign keys:
management.head_ID = head.head_ID
management.department_ID = department. Department_ID
Question:
How many heads of the departments are older than 56?
Gold SQL:
SELECT count(*) FROM head WHERE age > 56
```

A.2 Query #35 (medium)

```
Write only the MongoDB with no explanation for the query using the
following schema. Do not select extra columns that are not explicitly
requested.
Schema:
city(City_ID, Official_Name, Status, Area_km_2, Population,
  Census_Ranking)
farm(Farm_ID, Year, Total_Horses, Working_Horses, Total_Cattle, Oxen,
  Bulls, Cows, Pigs, Sheep_and_Goats)
farm_competition(Competition_ID, Year, Theme, Host_city_ID, Hosts)
competition_record(Competition_ID, Farm_ID, Rank)
Foreign keys:
farm_competition. Host_city_ID = city. City_ID
competition_record. Farm_ID = farm. Farm_ID
competition_record. Competition_ID = farm_competition. Competition_ID
```

```
Question:
Show the years and the official names of the host cities of competitions.
Gold SQL:
SELECT T2.Year, T1.Official_Name FROM city AS T1
JOIN farm_competition AS T2 ON T1.City_ID = T2.Host_city_ID
```

A.3 Query #86 (hard)

```
Write only the MongoDB with no explanation for the query using the
following schema. Do not select extra columns that are not explicitly
requested.
Schema:
Addresses(address_id, line_1, line_2, city, zip_postcode,
  state_province_county, country)
People(person_id, first_name, middle_name, last_name, cell_mobile_number,
  email_address, login_name, password)
Students(student_id, student_details)
Courses(course_id, course_name, course_description, other_details)
People_Addresses(person_address_id, person_id, address_id, date_from,
date_to)
Student_Course_Registrations(student_id, course_id, registration_date)
Student_Course_Attendance(student_id, course_id, date_of_attendance)
Candidates(candidate_id, candidate_details)
Candidate_Assessments(candidate_id, qualification, assessment_date,
  asessment_outcome_code)
Foreign keys:
Students.student_id = People.person_id
People_Addresses.address_id = Addresses.address_id
People_Addresses.person_id = People.person_id
Student_Course_Registrations.course_id = Courses.course_id
Student_Course_Registrations.student_id = Students.student_id
Student_Course_Attendance.student_id = Student_Course_Registrations.
  student_id
Student_Course_Attendance.course_id = Student_Course_Registrations.
  course_id
Candidates.candidate_id = People.person_id
Candidate_Assessments.candidate_id = Candidates.candidate_id
Question:
Find the cell mobile number of the candidates whose assessment code is
  'Fail'?
Gold SQL:
SELECT T3.cell_mobile_number FROM candidates AS T1
JOIN candidate_assessments AS T2 ON T1.candidate_id = T2.candidate_id
JOIN people AS T3 ON T1.candidate_id = T3.person_id WHERE
  T2.asessment_outcome_code = 'Fail'
```

A.4 Query #172 (extra hard)

```
Write only the MongoDB with no explanation for the query using the
following schema. Do not select extra columns that are not explicitly
requested.
```

```
Schema:
station(id, name, lat, long, dock_count, city, installation_date)
status(station_id, bikes_available, docks_available, time)
trip(id, duration, start_date, start_station_name, start_station_id,
  end_date, end_station_name, end_station_id, bike_id, subscription_type,
  zip_code)
weather(date, max_temperature_f, mean_temperature_f, min_temperature_f,
    max_dew_point_f,mean_dew_point_f, min_dew_point_f, max_humidity,
    mean_humidity, min_humidity,max_sea_level_pressure_inches,
    mean_sea_level_pressure_inches, min_sea_level_pressure_inches,
    max_visibility_miles, mean_visibility_miles, min_visibility_miles,
    max_wind_Speed_mph, mean_wind_speed_mph, max_gust_speed_mph,
    precipitation_inches, cloud_cover, events, wind_dir_degrees,
    zip_code)
Foreign keys:
status.station_id = station.id
Question:
What are names of stations that have average bike availability above 10
  and are not located in San Jose city?
Gold SQL:
SELECT T1.name FROM station AS T1 JOIN status AS T2 ON T1.id =
    T2.station_id
GROUP BY T2.station_id HAVING avg(bikes_available) > 10
EXCEPT SELECT name FROM station WHERE city = 'San Jose'
```

B. Prompt examples used in the experiments

Below are the sample queries for different difficulty levels: "*easy*," "*medium*," "*hard*," and *"extra hard."* These prompts are picked among 620 ground-truth queries mentioned in Table 1 (verified MQL queries generated from Spider dev dataset) and used in different experiments.

B.1 Query #181 (easy)

```
Write only the MongoDB with no explanation for the query using the
following schema. Do not select extra columns that are not explicitly
requested.
Schema:
airlines(uid, Airline, Abbreviation, Country)
airports(City, AirportCode, AirportName, Country, CountryAbbrev)
flights(Airline, FlightNo, SourceAirport, DestAirport)
Foreign keys:
flights. DestAirport = airports. AirportCode
flights. SourceAirport = airports. AirportCode
Question:
What country is Jetblue Airways affiliated with?
```

B.2 Query #268 (medium)

```
Write only the MongoDB with no explanation for the query using the
following schema. Do not select extra columns that are not explicitly
requested.
```

```
Schema:
employee(Employee_ID, Name, Age, City)
shop(Shop_ID, Name, Location, District, Number_products, Manager_name)
hiring(Shop_ID, Employee_ID, Start_from, Is_full_time)
evaluation(Employee_ID, Year_awarded, Bonus)
Foreign keys:
hiring. Employee_ID = employee. Employee_ID
hiring. Shop_ID = shop. Shop_ID
evaluation. Employee_ID = employee. Employee_ID
Question:
Find the number of shops in each location.
```

B.3 Query #381 (hard)

```
Write only the MongoDB with no explanation for the query using the
following schema. Do not select extra columns that are not explicitly
requested.
Schema:
Ref_Template_Types(Template_Type_Code, Template_Type_Description)
Templates(Template_ID, Version_Number, Template_Type_Code,
  Date_Effective_From, Date_Effective_To, Template_Details)
Documents(Document_ID, Template_ID, Document_Name, Document_Description,
  Other_Details)
Paragraphs(Paragraph_ID, Document_ID, Paragraph_Text, Other_Details)
Foreign keys:
Templates. Template_Type_Code = Ref_Template_Types. Template_Type_Code
Documents. Template_ID = Templates. Template_ID
Paragraphs. Document_ID = Documents. Document_ID
Question:
What are the ids of documents that contain the paragraph text 'Brazil'
  and 'Ireland'?
```

B.4 Query #517 (extra hard)

```
Write only the MongoDB with no explanation for the query using the
following schema. Do not select extra columns that are not explicitly
requested.
Schema:
Addresses(address_id, line_1, line_2, line_3, city, zip_postcode,
  state_province_county, country, other_address_details)
Courses(course_id, course_name, course_description, other_details)
Departments(department_id, department_name, department_description,
  other_details)
Degree_Programs(degree_program_id, department_id, degree_summary_name,
  degree_summary_description, other_details)
Sections(section_id, course_id, section_name, section_description,
  other_details)
Semesters(semester_id, semester_name, semester_description,
  other_details)
Students(student_id, current_address_id, permanent_address_id,
  first_name, middle_name, last_name, cell_mobile_number, email_address,
  ssn, date_first_registered, date_left, other_student_details)
Student_Enrolment(student_enrolment_id, degree_program_id, semester_id,
  student_id, other_details)
```

```
Student_Enrolment_Courses(student_course_id, course_id,
  student_enrolment_id)
Transcripts(transcript_id, transcript_date, other_details)
Transcript_Contents(student_course_id, transcript_id)
Foreign keys:
Degree_Programs.department_id = Departments.department_id
Sections.course_id = Courses.course_id
Students.permanent_address_id = Addresses.address_id
Students.current_address_id = Addresses.address_id
Student_Enrolment.student_id = Students.student_id
Student_Enrolment.semester_id = Semesters.semester_id
Student_Enrolment.degree_program_id = Degree_Programs.degree_program_id
Student_Enrolment_Courses.student_enrolment_id = Student_Enrolment.
  student_enrolment_id
Student_Enrolment_Courses.course_id = Courses.course_id
Transcript_Contents.transcript_id = Transcripts.transcript_id
Transcript_Contents.student_course_id = Student_Enrolment_Courses.
  student_course_id
Question:
What is the name and id of the department with the most number of
  degrees ?
```

C. Nested query experiment

C.1 Hand-crafted queries

Below are the hand-crafted queries for this experiment:

```
1. Find the listing ids and names in Istanbul with hosts that their
government id is verified
2. Find the listing ids and names in Istanbul with hosts that their
facebook account is verified
3. Show the listing ids and names in Brazil with the exact location
unknown
4. Show the listing ids and names with free parking and hair dryer in
Porto, with at least 100 reviews
5. Show the listing ids and names in Montreal with a flexible
cancellation policy
6. Find listing ids and names with at least 3 bedrooms in Barcelona
7. Find pet-friendly listing ids and names in Sydney with a rating of 95
or above
8. Show the listing ids and names in New York with a minimum stay of 3
nights or less
9. Find listing ids and names in Rio De Janeiro with at least 2 bathrooms
10. Show the listing ids and names with a sea view in Istanbul
11. Find listing ids and names in Montreal with review score for
cleanliness over 8
12. Show the listing ids and names in New York with hosts that offer
breakfast
13. Find listing ids and names in Hong Kong with review score for
communication over 9 and offers Wifi and Cable TV
```

```
14. List listing ids with names which have private rooms in New York with
hosts that verified email
15. List listing ids with names which have shared rooms in New York with
a flexible cancellation policy
16. Find listing ids with names which have private rooms that requires
minimum stay of 3 nights in New York
17. Find listing ids with names in Hong Kong with no reviews from the
customers
18. Show the listing ids with names which have shared rooms with couch
type beds in Barcelona
19. Show listing ids with names which have private rooms that are cheaper
than 1000 dollars in New York
20. Find the listing ids and names in Istanbul with hosts that provide
Wifi
```

C.2 Prompt structure

Here is the generic prompt structure that is used for this experiment:

```
Write only the MongoDB with no explanation for the query using the
following schema.
Do not select extra columns that are not explicitly requested.
There are column names that have dot character in them, they represent
nested document structure.
For nested columns, wrap column name with double quote characters while
using them in the MongoDB query.

Schema:
      listings_and_reviews(listing_url, name, summary, space,
      description, neighborhood_overview, notes, transit, access,
      interaction, house_rules, property_type, room_type, bed_type,
      minimum_nights, maximum_nights, cancellation_policy, last_scraped,
      calendar_last_scraped, first_review, last_review, accommodates,
      bedrooms, beds, number_of_reviews, bathrooms, amenities, price,
      extra_people, guests_included, images.thumbnail_url,
      images.medium_url, images.picture_url, images.xl_picture_url,
      host.host_id, host.host_url, host.host_name, host.host_location,
      host.host_about, host.host_thumbnail_url, host.host_picture_url,
      host.host_neighbourhood, host.host_is_superhost,
      host.host_has_profile_pic, host.host_identity_verified,
      host.host_listings_count, host.host_total_listings_count,
      host.host_verifications, address.street, address.suburb,
      address.government_area, address.market, address.country,
      address.country_code, address.location.type,
      address.location.coordinates, address.location.is_location_exact,
      availability.availability_30, availability.availability_60,
      availability.availability_90, availability.availability_365,
      review_scores.review_scores_accuracy,
      review_scores.review_scores_cleanliness, review_scores.review_
      scores_checkin,
      review_scores.review_scores_communication, review_scores.review_
      scores_location, review_scores.review_scores_value,
      review_scores.review_scores_rating,
```

```
      reviews.date, reviews.listing_id, reviews.reviewer_id,
      reviews.reviewer_name, reviews.comments)
Question:
Find the listing ids and names in Istanbul with hosts that their
  government id is verified
```

C.3 Base model outputs

```
- Find the listing ids and names in Istanbul with hosts that their
government id is verified
db.listings_and_reviews.find( { "address.market": "Istanbul", "host.
  host_identity_verified": true }, { "listing_id": 1, "name": 1 } )
-------
+ Find the listing ids and names in Istanbul with hosts that their
facebook account is verified
db.listings_and_reviews.find( { "address.market": "Istanbul", "host.
host_verifications": /.*facebook.*/i }, { "listing_id": 1, "name": 1 })
-------
+ Show the listing ids and names in Brazil with the exact location
unknown
db.listings_and_reviews.find( {"address.country": "Brazil",
  "address.location.is_location_exact": false}, {_id: 0, "listing_id":
  1, "name": 1} )
-------
- Show the listing ids and names with free parking and hair dryer in
Porto, with at least 100 reviews
db.listings_and_reviews.find({ "amenities": { $all: ["Free parking",
  "Hair dryer"] }, "address.market": "Porto", "number_of_reviews":
  { $gte: 100 } }, { "listing_id": 1, "name": 1 })
-------
+ Show the listing ids and names in Montreal with a flexible cancellation
policy
db.listings_and_reviews.find({"address.market": "Montreal",
  "cancellation_policy": "flexible"}, {"listing_id": 1, "name": 1,
  "_id": 0})
-------
+ Find listing ids and names with at least 3 bedrooms in Barcelona
db.listings_and_reviews.find({bedrooms: {$gte: 3}, "address.market":
  "Barcelona"}, {listing_id: 1, name: 1})
-------
- Find pet-friendly listing ids and names in Sydney with a rating of 95
or above
{ "property_type": "Apartment", "address.market": "Sydney",
  "amenities": {"$regex": ".*pet.*", "$options": "i"},
  "review_scores.review_scores_rating": {"$gte": 95} }
-------
+ Show the listing ids and names in New York with a minimum stay of 3
nights or less
db.listings_and_reviews.find( { "address.market": "New York",
  "minimum_nights": { $lte: 3 } }, { "listing_id": 1, "name": 1 } )
-------
+ Find listing ids and names in Rio De Janeiro with at least 2 bathrooms
```

```
db.listings_and_reviews.find( { "address.market": "Rio De Janeiro",
  "bathrooms": {"$gte": 2} }, { "listing_url": 1, "name": 1 } )
-------
- Show the listing ids and names with a sea view in Istanbul
db.listings_and_reviews.find( { "address.market": "Istanbul",
  "description": /sea view/i }, { "listing_id": 1, "name": 1 } )
-------
+ Find listing ids and names in Montreal with review score for
cleanliness over 8
db.listings_and_reviews.find( { "address.market": "Montreal",
  "review_scores.review_scores_cleanliness": { $gt: 8 } }, { "name": 1,
  "reviews.listing_id": 1 } );
-------
- Show the listing ids and names in New York with hosts that offer
breakfast
{ "address.market": "New York", "amenities": { "$regex": "breakfast" } }
-------
+ Find listing ids and names in Hong Kong with review score for
communication over 9 and offers Wifi and Cable TV
db.listings_and_reviews.find({ "address.market": "Hong
  Kong", "amenities": {$all: ["Wifi", "Cable TV"] },
  "review_scores.review_scores_communication": { $gt: 9 } }, {
  "listing_id": 1, "name": 1 })
-------
+ List listing ids with names which have private rooms in New York with
hosts that verified email
db.listings_and_reviews.find( { "room_type": "Private room",
  "address.market": "New York", "host.host_verifications": /email/i },
  { "name": 1, "listing_url": 1 } )
-------
- List listing ids with names which have shared rooms in New York with a
flexible cancellation policy
{ "property_type": "Shared room", "address.market": "New York",
  "cancellation_policy": "flexible" }
-------
- Find listing ids with names which have private rooms that requires
minimum stay of 3 nights in New York
db.listings_and_reviews.find({ "name": /private room.*new york/i, "room_
  type": "Private room", "minimum_nights": 3 }, { "listing_id": 1 })
-------
- Find listing ids with names in Hong Kong with no reviews from the
customers
db.listings_and_reviews.find({"neighborhood_overview": "Hong Kong",
  "number_of_reviews": {$eq: 0}}, {"listing_id": 1, "name": 1})
-------
- Show the listing ids with names which have shared rooms with couch type
beds in Barcelona
db.listings_and_reviews.find({ "bed_type": "Couch", "room_type": "Shared
  room", "address.market": "Barcelona" }, { "listing_id": 1, "name": 1 })
-------
```

```
- Show listing ids with names which have private rooms that are cheaper
than 1000 dollars in New York
db.listings_and_reviews.find({"room_type": "Private room", "price":
  { $lt: 1000 }, "address.government_area": "New York"},{"listing_id":
  1,"name": 1})
-------
- Find the listing ids and names in Istanbul with hosts that provide Wifi
db.listings_and_reviews.find( { "host.host_location": "Istanbul",
  "amenities": { $regex: /wifi/i } }, { "listing_id": 1, "name": 1,
  "_id": 0 } )
9/20 correct queries
```

C.4 Fine-tuned model outputs

```
- Find the listing ids and names in Istanbul with hosts that their
government id is verified
db.listings_and_reviews.find( { "address.market": "Istanbul", "host.host_
  identity_verified": true, "host.host_total_listings_count":
  { $gte: 1 } }, { _id: 0, name: 1, "host.host_id": 1 } )
-------
+ Find the listing ids and names in Istanbul with hosts that their
facebook account is verified
db.listings_and_reviews.find( { "address.market": "Istanbul",
  "host.host_verifications": /.*facebook.*/, "host.host_identity_
  verified": true }, { _id: 0, name: 1, "host.host_id": 1 } )
-------
+ Show the listing ids and names in Brazil with the exact location
unknown
db.listings_and_reviews.find( { "address.country": "Brazil",
  "address.location.is_location_exact": { $ne: true } }, { "name": 1,
  "listing_id": 1, "_id": 0 } )
-------
- Show the listing ids and names with free parking and hair dryer in
Porto, with at least 100 reviews
db.listings_and_reviews.find( { $and: [ { " address.market": "Porto"
  }, { "amenities": { $all: [/"Free parking"/, /"Hair dryer"/] } },
  { "number_of_reviews": { $gte: 100 } } ] }, { "listing_id": 1, "name":
  1, "_id": 0 } )
-------
+ Show the listing ids and names in Montreal with a flexible cancellation
policy
db.listings_and_reviews.find( { "address.market": "Montreal",
  "cancellation_policy": /flexible/i }, { _id: 0, listing_id: 1, name:
  1 } )
-------
+ Find listing ids and names with at least 3 bedrooms in Barcelona
db.listings_and_reviews.find( { "address.market": "Barcelona",
  "bedrooms": { $gte: 3 } }, { "listing_id": 1, "name": 1, "_id": 0 })
-------
- Find pet-friendly listing ids and names in Sydney with a rating of 95
or above
```

```
db.listings_and_reviews.find( { address: { $regex: /sydney/i } }, { name:
  1, _id: 0, listing_id: 1 } ) .sort({ "review_scores_rating": -1 })
-------
+ Show the listing ids and names in New York with a minimum stay of 3
nights or less
db.listings_and_reviews.find( { "address.market": "New York",
  minimum_nights: { $lte: 3 } }, { name: 1, listing_id: 1, _id: 0 } )
-------
+ Find listing ids and names in Rio De Janeiro with at least 2 bathrooms
db.listings_and_reviews.find( { "address.market": "Rio De Janeiro",
  "bathrooms": { $gte: 2 } }, { listing_id: 1, name: 1, _id: 0 } )
-------
+ Show the listing ids and names with a sea view in Istanbul
db.listings_and_reviews.find( { $and: [ { "address.market": "Istanbul" },
  { amenities: /sea view/i } ] }, { name: 1, "address.market": 1 } )
-------
+ Find listing ids and names in Montreal with review score for
cleanliness over 8
db.listings_and_reviews.find( { "address.market": "Montreal",
  "review_scores.review_scores_cleanliness": { $gt: 8 } }, { _id: 0,
  "reviews.listing_id": 1, name: 1 } )
-------
+ Show the listing ids and names in New York with hosts that offer
breakfast
db.listings_and_reviews.find( { $and: [ { "address.market": "New York" },
  { amenities: { $regex: /breakfast/, $options: 'i' } } ] }, { name: 1,
  _id: 0, listing_id: 1 } } )
-------
+ Find listing ids and names in Hong Kong with review score for
communication over 9 and offers Wifi and Cable TV
db.listings_and_reviews.find( { "address.market": "Hong Kong",
  "review_scores.review_scores_communication": { $gte: 9 },
  "amenities": { $regex: /wifi/i, $regex: /cable tv/i } }, { "name": 1,
  "reviews.listing_id": 1, "_id": 0 } )
-------
- List listing ids with names which have private rooms in New York with
hosts that verified email
db.listings_and_reviews.find( { "bedrooms": 1, "address.market":
  "newyorkcity", "host.host_email_verified": "t" }, { "name": 1,
  "listing_id": 1, "_id": 0 } )
-------
- List listing ids with names which have shared rooms in New York with a
flexible cancellation policy
db.listings_and_reviews.find( { "bed_type": "Real Bed", "address.market":
  "New York", "room_type": "Shared room", "cancellation_policy":
  "flexible" }, { "name": 1 } )
-------
+ Find listing ids with names which have private rooms that requires
minimum stay of 3 nights in New York
```

```
db.listings_and_reviews.find( { "name": { "$regex": /private room/ },
  "minimum_nights": { "$gte": 3 }, "address.market": "New York" }, {
  "listing_id": 1, "_id": 0 } )
-------
- Find listing ids with names in Hong Kong with no reviews from the
customers
db.listings_and_reviews.aggregate([ { $match: { "address.market": "Hong
  Kong", "reviews": { $exists: false } }, }, { $project: { _id: 0,
  listing_id: 1 } } ])
-------
- Show the listing ids with names which have shared rooms with couch type
beds in Barcelona
db.listings_and_reviews.find( { "bed_type": "Real Bed", "property_type":
  /Shared room/, "address.market": "Barcelona" }, { "name": 1,
  "listing_id": 1, "_id": 0 } )
-------
+ Show listing ids with names which have private rooms that are cheaper
than 1000 dollars in New York
db.listings_and_reviews.find( { $and: [ {"address.market": "New York"},
  {"price": { $lt: 1000 }}, {"room_type": "Private room"} ] }, { "name":
  1, "_id": 0 } )
-------
+ Find the listing ids and names in Istanbul with hosts that provide Wifi
db.listings_and_reviews.find( { $and: [ { "address.market": "Istanbul" },
  { amenities: /wifi/i } ] }, { listing_id: 1, name: 1, _id: 0 } )
13/20 correct queries
```

D. Fine-Tuning Loss Graphs

According to the loss graphs, which are directly taken from the model outputs, and given in Figure 7, both train and validation losses decrease gradually which indicate that the fine-tuned models do not suffer from overfitting.

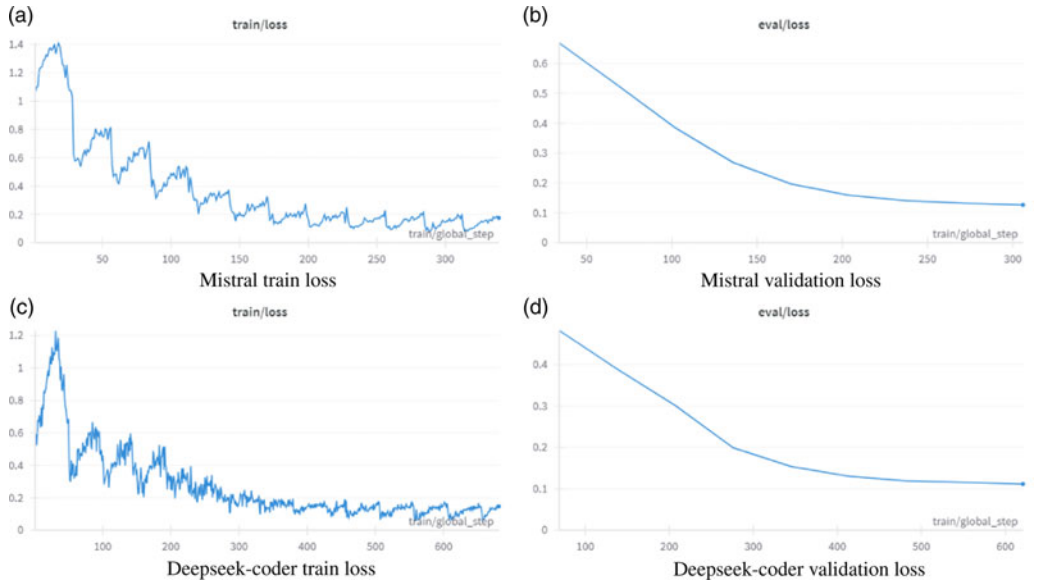

Figure 7. Fine-tuning train-val loss graphs.

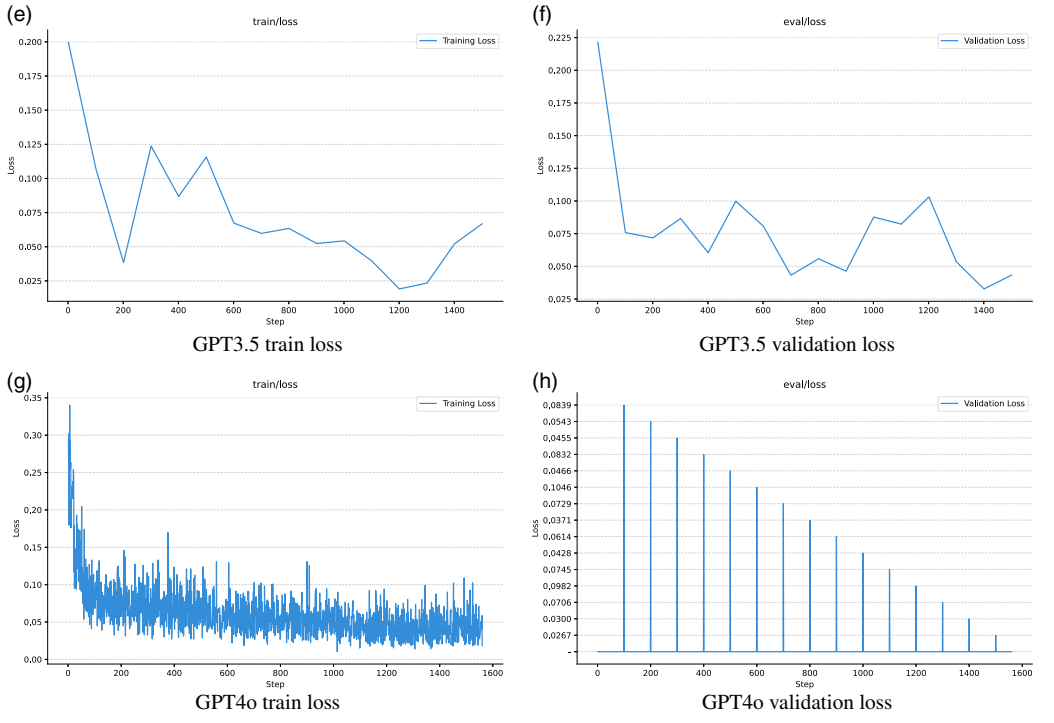

Figure 7. (Continued)

Cite this article: Özer AG, Cekinel RF, Toroslu IH and Karagoz P. DocSpider: a dataset of cross-domain natural language querying for MongoDB. *Natural Language Processing* https://doi.org/10.1017/nlp.2024.63