2024 IEEE Latin American Conference on Computational Intelligence (LA-CCI) | 979-8-3503-7457-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/LA-CCI62337.2024.10814762

Comparison of LLM Models and Strategies for Structured Query Construction from Natural Language Queries

Franklin Cardeñoso Fernández
PUC-Rio
Rio de Janeiro, Brazil
Email: franklin@aluno.puc-rio.br

Robinson Luiz Souza Garcia
Petrobras
Rio de Janeiro, Brazil
robinson.garcia@petrobras.com.br

Wouter Caarls
PUC-Rio
Rio de Janeiro, Brazil
Email: wouter@puc-rio.br

***Abstract*—The advent of large language models (LLM) has sharply increased the power of machine translation because of their remarkable success in many text processing tasks. This feature makes LLMs attractive options for specific tasks, such as translating natural language queries into keyword-based queries. Within this research, we compare the different techniques applied in LLMs for open and closed-source model options to facilitate decision-making for researchers and practitioners. Our findings present that while fine-tuning is a highly recommended option, prompting techniques allow us to achieve similar performance with commercial tools; however, before any model selection, it is essential to consider the amount of data, computational resources, and data privacy.**

I. Introduction

Translation tasks have gained significant attention for many years, including in translating human natural language queries (NLQ) into structured text or queries. This particular attention relies on the fact that structured queries are helpful when users interact in web applications with search engines, where particular query formats, such as keyword-based queries (KQ), are commonly required to interact with databases and retrieve the desired information.

Translating queries from human intentions can be challenging, especially if the targeted query format is too particular. In this context, different solutions have been proposed for translating NLQs to specific formats, such as Kusto queries [1], corpus queries [2], OverpassQL queries [3], and SQL queries, for example, where many solutions were proposed given its popularity [4], [5], [6], [7]. Unlike these formats, a specific case of text translation is the construction of keyword-based queries (KQs) from natural language text. This task becomes particularly problematic because often the exact words used in the underlying database may not be present in the input sentences, or incomplete sentences could be used as user queries. While this task has been addressed for years with traditional approaches [8], [9], [10], [11], the use of large language models (LLM) appears as an exciting option because of their remarkable success in various text processing tasks [12].

979-8-3503-7457-5/24/$31.00 ©2024 IEEE

However, the emergence of different open-source and closed-source models makes selecting a particular tool or approach to address this task problematic. While some allow fine-tuning with lower computational resources, other options allow impressive results by applying prompting techniques, with every option having its advantages and risks.

This research presents a comparative analysis of using LLMs to address the problem of translating NLQs into KQs. We conduct this comparison at different complexity levels to observe how the different techniques respond to the task. Our comparison involves different approaches, including zero-shot, few-shot, RAG, and fine-tuning (when possible) for two different model types: proprietary and open-source variants. These comparisons allow us to gain a better understanding of the performance of these techniques and models when proprietary data is involved.

We structured this paper as follows: Section 2 presents related works and briefly describes them, and then we provide the conceptual preliminary concepts in Section 3. Next, we describe the problem and the experimental methodology we followed in Section 4, while the results and their corresponding discussion are presented in Sections 5 and 6, respectively. Finally, we present the conclusions in Section 7, proposing new guidelines for future work.

II. Related work

Several studies have been performed on the task of translating from natural language queries (NLQ) to structured representations. One of them is the one developed by [13], which proposes the use of controlled natural language to be translated into a formal language, such as SPARQL, Sparklis, or SQUALL, for instance. Furthermore, [2] and [3] address the problem of translating from text to corpus query language and text to OverpassQL queries, respectively. While the first approach is built on top of the BART model for fine-tuning and GPT-4 for few-shot to perform the evaluation, the second approach evaluates fine-tuning with the T5 LLM model family. In the same context, NL2KQL is a framework to convert NLQs to Kusto Query Language (KQL), which uses RAG and few-shot learning at its core [1].

On the other hand, translating NLQ to SQL queries is probably one of the most popular applications for LLMs in translation tasks. Among the developed research, we can find the Zero-shot NL2SQL framework, which combines the knowledge of T5-3B model to generate SQL candidates and GPT-3.5-turbo to refine the responses and obtain the final SQL queries [4]. Similarly, [5] presents a divide-and-conquer framework that uses T5 to create SQL structures combined with PICARD model to populate placeholders from these structures with concrete values. In the same vein, ODIS is a framework that uses in-context learning by providing demonstrations from hybrid sources and compares open-source and closed-source LLMs [6]. Furthermore, these authors also present a comparison of prompt strategies such as zero-shot and few-shot with GPT-3 Codex and ChatGPT [7].

More specifically, the task of translating NLQ to KQ has been addressed by [11] by applying recurrent neural networks to generate KQs that preserve the meaning of the original query. Unlike this previous work, our approach evaluates the use of LLMs to address this task by generating KQs that contain specific keywords from the original database.

III. Preliminaries

A. Foundational and large language models

Roughly speaking, foundational models (FMs) are large deep-learning architectures trained with massive datasets of generalized information to generate outputs from input prompts in the form of human language instructions [14].

In this context, large language models (LLMs) are a particular subset of FMs designed to understand and generate natural language [15]. The massive corpus of text used during training allows them to learn grammar, semantics and conceptual relationships to infer from context and generate coherent and contextual responses from a wide range of text processing tasks.

1) Generative Pre-training Transformer (GPT): GPT-*n* is probably the most well-known family of closed-source LLM models. Developed by OpenAI, these models can generate human-like natural language text from text inputs (or images, in recent releases). Between the most known versions, we can find GPT-3.5 Turbo (GPT-3.5 chat optimized), or GPT-4 [12]. Although there is no official disclosure about the model size of recent models, it is worth mentioning that GPT-3 has 175 billion parameters and was trained on 500 billion tokens of training data [16].

2) Mistral-7B: This popular open-source LLM model released in September 2023 implements grouped-query attention (GQA) modules and sliding window attention (SWA) in its architecture, increasing the speed for inference time and more extended text sequence management. Because of that, this model was able to outperform other models in its size categories, such as Llama 1 and 2 or CodeLlama, for example, when compared in different benchmarks [17].

B. Zero-shot and Few-shot learning

Zero-shot and few-shot learning, in the context of LLMs, are prompt techniques that allow the exploitation of the models' skills for pattern recognition acquired during training to be used at inference time to adapt the model to perform a desired task [16]. While zero-shot learning only provides the instructions itself, for few-shot learning, some examples are also supplied within the text input. Thereby, the model is adapted to recognise and perform the targeted task [18].

C. Retrieval Augmented Generation (RAG)

RAG is a method that combines a pre-trained retriever with a pre-trained FM to produce more accurate and contextually relevant responses [19].Within a standard RAG scheme, first, the specialised data, also called external data, is embedded and stored in a vector database; then, every time a user performs a query, the vector representation of the user query is matched with the database to retrieve the most relevant documents from it by using a similarity metric. Next, the initial prompt is enhanced by adding the relevant retrieved data in context through prompt engineering techniques. Finally, the generation model uses the enhanced and contextualised prompt to produce a coherent and informative response [20].

D. Fine-tuning

Fine-tuning LLMs is a supervised learning process that involves adapting an FM to a specific domain by training it with a smaller and task-specific dataset to adjust its parameters for the target task [21]. Given the complexity of the actual models, fine-tuning full LLMs is not recommended since retraining larger models requires more computational resources. However, the introduction of model quantisation or approaches such as PEFT [22] or QLoRA [23], for example, makes it possible to fine-tune LLMs with lower computational resources by training only adapter modules.

IV. Experimental methodology

A. Problem overview

In this work, we focus on translating natural language queries into keyword-based queries. We do this to improve interactions between users and the search engine and retrieve proprietary data[1] more effectively.

The base system, described in [24], allows users to interact with an information retrieval system by writing KQs and receiving related information in a tabular format. These keywords are divided into three categories: `classes`, `properties`, and `values`. One issue with the current system is that users who are not familiar with it may struggle to get relevant results since some database classes have multiple common names. Furthermore, some classes may have identically named properties, requiring the class to be specified before each property.

[1]Due to confidentiality, descriptions or details will not be explicitly shown. The language of the data is Portuguese.

Therefore, we aim to introduce a new feature that improves the system by enabling users to input NLQs, which will be translated into KQs that the system can directly use, simplifying user interaction with the platform. For instance:

- **Input:** Situation of the objects on the FPPLB platform
- **Output:** Object Platform situation Name = FPPLB

Consequently, given that the intention is to implement an additional LLM-based layer on top of this existing search system, we aimed to test different approaches such as zero-shot learning, few-shot learning, RAG, and fine-tuning to develop this feature. The results of this comparison are presented in this paper.

B. Synthetic dataset generation

To start with this process, first, we narrowed down the search topics by manually selecting the most common queries and then incorporating suggestions from expert users to create a set of potential queries that regular users may use in the system. Next, we created a basic dataset of KQs using input/output keyword-based examples by cleaning the labels to ensure that they provide the expected information and refining the inputs by using different possible textual descriptions.

After testing and validating the preliminary KQs, we used a GPT-4 model in a few-shot fashion to generate the corresponding NLQ inputs. To further diversify the dataset, we requested the GPT-4 model to produce NLQs in various styles, including verbosity, simplicity, technicality, and detail.

Once the dataset was created, the output KQs queries were curated at different complexity levels to generate three datasets: NL2CLS, NL2PROP, and NL2CPV. Their description is provided below:

- **NL2CLS:** Dataset with NLQs as inputs and KQs with only `classes` as outputs.

Input:	Show the orders opened with a lack of material at P-74
Output:	order platform

- **NL2PROP:** Dataset with NLQs as inputs and KQs including `classes` and `properties`.

Input:	Show the orders opened with a lack of material at P-74
Output:	order situation platform

- **NL2CPV:** Dataset with NLQs as inputs and KQs including `classes`, `properties` and `values` as outputs.

Input:	Show the orders opened with a lack of material at P-74
Output:	order situation=open 'system status' contains MATF Platform=P-74

C. Implemented systems

For this research, we decided to employ both open-source and closed-source LLM models. Mistral-7B in its two variants (base and instruct) was chosen for the open-source option, and GPT-3.5 Turbo and GPT-4 for the closed-source alternative. With these models, we performed the following approaches: zero-shot, few-shot, RAG, and fine-tuning; however, we determined the following additional considerations: We tested GPT-3.5, GPT-4, and Mistral-7B-Instruct for the zero-shot, few-shot, and RAG, while only the two variants of Mistral were considered for the fine-tuning process.

TABLE I
PROMPT TEMPLATE

You are a powerful content extractor whose job is to create queries on KEYWORDS provided from a natural language question. Use the following criteria:
< INSTRUCTIONS >
1. identify the natural language query.
2. Extract the keywords from the query concerning the context provided.
###############
Given the following context.
< CONTEXT >
###############
Ensure you only return the generated ###Answer and don't say anything else.
###############
To help you with your task, some EXAMPLES are provided.
< EXAMPLES >
##Question:< QUERY >
##Answer

1) Zeroshot learning: The implementation of zero-shot learning varied according to the dataset type addressed. We started with a prompt template like the one presented in Table I. Then, for the case of the NL2CLS dataset, we used the prompt with the set of instructions and appending a detailed list of the `classes` and their respective descriptions as the < CONTEXT >. Finally, we performed a curation process to clean the generated outputs and maintain only the detected objective keywords according to a pre-defined list.

For the NL2PROP dataset, we performed a two-staged process, first, an initial stage was done by following the process described in the previous paragraph. Next, after the outputs were cleaned, a new prompt was built for each detected class by appending the original user query, the original instructions and replacing < CONTEXT > with specific information about the class properties and their descriptions for the given class. Finally, the generated information was cleaned and appended to return the final KQ.

Finally, for the NL2CPV dataset, the prompt was built by including the instructions, the user query, and all the possible information about the class, properties, and possible values as the context. Then, the generated output was curated to maintain only the relevant information. We applied these operations to all the tested models.

2) Fewshot learning: Given that few-shot learning provides expert demonstrations of how to perform the task, a unique prompt structure was built with the prompt template for the three dataset types, where a set of 20 examples was carefully selected from the respective datasets. This prompt contains the instructions, information about the classes and properties, and, finally, the demonstrations. We applied these operations to all the tested models.

3) RAG: We built a vector database for the RAG implementation using the ChromaDB engine[2], embedding the queries from the training dataset and storing the resulting vectors into the vector database. Similar to the few-shot learning case, in the RAG approach, a unique prompt was built in order to attend to all the dataset types. This prompt comprises instructions, information about the classes and properties, and samples of how to perform the task. Then, every time a user query was called, we selected the 20 more similar samples to the user query to be appended in the examples prompt section. This process was the same for all the models and datasets tested.

4) Fine-tuning: Finally, for the fine-tuning process, we used the QLoRA approach to fine-tune all the linear layers of the 4-bit quantized version of the models. We used the following parameters for QLoRA: the rank is 8, the scaling factor (alpha) is 16, and the dropout probability is 0.1. Furthermore, the prompt built contains the instructions section and asks almost directly to the model to translate the NLQ input and return the generated KQ query output.

5) Evaluation metrics: For all the tested combinations, we use statistical automatic metrics such as ROUGE-L [25], METEOR [26], COMET [27] and Exact Match (EM); as well as the LLM-based BERTScore (BS) metric [28].

D. Experimental setup

All the implementations concerning the Mistral-7B models were done in a desktop workstation with a 24GB GeForce RTX 3090 graphic card with Ubuntu 20.04.6 LTS as the operating system and Python 3.10 as the programming language using the `transformers` package. On the other hand, for the GPT-*n* models, a private workspace was used to consume the Azure OpenAI Service through REST API access.

V. Results

Following the established guidelines, different experiments were carried out. Tables II, III, and IV summarize our findings for the NL2CLS, NL2PROP, and NL2CPV datasets, respectively. Furthermore, within these tables, we also summarize the results for the different approaches such as zero-shot, few-shot, RAG, and fine-tuning tested in GPT-3.5 Turbo, GPT-4, Mistral-base, and Mistral Instruct, respectively. Finally, some output samples are presented in Tables V, VI and VII.

VI. Discussion

From all the presented results, we can draw different insights. First, looking at the average accuracy, an important result for all the datasets is the superior performance for the fine-tuning approach for both the Mistral-base and the Mistral-instruct models compared to the remaining models. However, we can not observe a significant performance difference between both models for the three datasets, practically achieving the same performance with both versions of Mistral in the fine-tuning approach.

[2]https://docs.trychroma.com

Another notable result is the performance of the GPT-4 model, which consistently outperforms the other models in the non-finetuning scenarios. It is only outperformed by the GPT-3.5 Turbo model in the NL2CLS dataset with the zero-shot method. Additionally, when combined with the RAG method, GPT-4 surpasses fine-tuning for the NL2CPV dataset, achieving similar performance to the fine-tuned Mistral models in the remaining datasets.

Overall, the GPT-4 model demonstrates superior performance when prompting techniques are applied, followed by the GPT-3.5 Turbo model and Mistral. This is expected, as GPT-*n* models have more trained parameters than Mistral-7B models, making them more effective tools when fine-tuning is not required. On the other hand, if we observe the overall performance of the techniques, another common result is the superior performance of fine-tuning, followed by RAG, few-shot and zero-shot at the final.

For the zero-shot prompting method, we can observe that while for the NL2CLS dataset, all models achieved reasonable results, this performance decreases for the NL2PROP dataset, this could probably be produced by the two-stage technique used in this case, which indicates that although we can obtain a specific performance when a smaller space of keywords domain is targeted, their isolated use in custom prompts to retrieve the properties affects the performance, making this practice not too recommended. This affirmation can be supported by the results in the NL2CPV dataset, which outperforms the previous case, where the used prompt contains all the possible context about the classes, properties, and their respective descriptions.

Surprisingly, if we look at the average results for RAG and few-shot in the NL2CLS dataset, we can observe that Mistral-7B Instruct outperforms the performance of GPT-3.5 Turbo; this is an interesting behaviour because it could indicate that Mistral model can understand and perform simpler tasks better than GPT-3.5. However, while the performance of Mistral decreases consistently when the task complexity increases, a contrary effect happens with GPT-3.5. These results can be contrasted with the output samples presented in Tables V, VI and VII.

VII. Conclusion and future work

Within this research, we presented a comparison between different techniques such as zero-shot, few-shot, RAG, and fine-tuning for three different models among open and closed-source such as GPT-3.5, GPT-4, and Mistral-7B. We addressed the translation task from NLQs to KQs from a database with proprietary data that contains three levels of data: classes, properties, and values; by using this database, we built three types of dataset by varying the output labels: one containing only classes, the second which contains classes and properties and the third that contains the classes, properties, and values.

For our case, among our findings, fine-tuning is a recommended practice since a small model such as Mistral-7B in both its variants (base and instruct) could outperform practically all the targeted output types. However, we could not

TABLE II
RESULTS FOR NL2CLS DATASET

	Zero-shot (ZS)			Few-shot (FS)			RAG			Fine-tuning	
	GPT-3.5 Turbo	GPT-4	Mistral Instruct	GPT-3.5 Turbo	GPT-4	Mistral Instruct	GPT-3.5 Turbo	GPT-4	Mistral Instruct	Mistral Instruct	Mistral Base
ROUGE	0.681	0.688	0.627	0.733	0.841	0.794	0.733	0.953	0.903	0.987	0.987
EM	0.408	0.37	0.317	0.405	0.585	0.538	0.534	0.923	0.813	0.949	0.948
METEOR	0.458	0.463	0.45	0.663	0.726	0.629	0.708	0.818	0.762	0.839	0.84
COMET	0.8	0.795	0.786	0.837	0.904	0.873	0.836	0.956	0.933	0.977	0.977
BS (P)	0.89	0.892	0.83	0.896	0.95	0.925	0.885	0.98	0.942	0.995	0.995
BS (R)	0.863	0.863	0.81	0.916	0.95	0.92	0.927	0.985	0.947	0.995	0.995
BS (F1)	0.875	0.876	0.819	0.904	0.949	0.922	0.904	0.982	0.944	0.995	0.995
Average	**0.711**	0.707	0.663	0.765	**0.844**	0.8	0.79	**0.942**	0.892	**0.962**	**0.962**

TABLE III
RESULTS FOR NL2PROP DATASET

	Zero-shot (ZS)			Few-shot (FS)			RAG			Fine-tuning	
	GPT-3.5 Turbo	GPT-4	Mistral Instruct	GPT-3.5 Turbo	GPT-4	Mistral Instruct	GPT-3.5 Turbo	GPT-4	Mistral Instruct	Mistral Instruct	Mistral Base
ROUGE	0.351	0.429	0.331	0.807	0.83	0.748	0.92	0.975	0.889	0.976	0.974
EM	0.063	0.075	0.02	0.455	0.567	0.42	0.721	0.882	0.716	0.903	0.895
METEOR	0.311	0.377	0.307	0.797	0.813	0.71	0.928	0.962	0.859	0.966	0.963
COMET	0.578	0.619	0.568	0.861	0.894	0.842	0.933	0.964	0.916	0.967	0.966
BS (P)	0.734	0.774	0.686	0.921	0.952	0.909	0.964	0.99	0.942	0.992	0.991
BS (R)	0.768	0.797	0.73	0.934	0.947	0.908	0.975	0.991	0.942	0.991	0.991
BS (F1)	0.747	0.782	0.705	0.927	0.949	0.908	0.969	0.99	0.942	0.992	0.991
Average	0.507	**0.551**	0.478	0.815	**0.85**	0.778	0.916	**0.965**	0.887	**0.97**	0.967

TABLE IV
RESULTS FOR NL2CPV DATASET

	Zero-shot (ZS)			Few-shot (FS)			RAG			Fine-tuning	
	GPT-3.5 Turbo	GPT-4	Mistral Instruct	GPT-3.5 Turbo	GPT-4	Mistral Instruct	GPT-3.5 Turbo	GPT-4	Mistral Instruct	Mistral Instruct	Mistral Base
ROUGE	0.616	0.733	0.439	0.813	0.859	0.67	0.943	0.98	0.816	0.969	0.967
EM	0.108	0.236	0.0	0.425	0.546	0.209	0.712	0.856	0.449	0.814	0.806
METEOR	0.471	0.63	0.329	0.763	0.8	0.605	0.939	0.961	0.791	0.947	0.944
COMET	0.734	0.829	0.595	0.865	0.889	0.775	0.938	0.954	0.852	0.948	0.947
BS (P)	0.827	0.901	0.732	0.932	0.956	0.859	0.978	0.992	0.911	0.989	0.988
BS (R)	0.841	0.901	0.773	0.93	0.945	0.873	0.981	0.991	0.935	0.988	0.987
BS (F1)	0.833	0.898	0.75	0.931	0.95	0.865	0.979	0.991	0.922	0.988	0.988
Average	0.633	**0.732**	0.517	0.808	**0.849**	0.694	0.924	**0.961**	0.811	**0.949**	0.947

TABLE V
OUTPUT SAMPLE FOR NL2CLS DATASET

NLQ:	**"Show the orders opened with a lack of material at P-74"**
KQ (Ref):	**"order platform"**
ZS GPT-3.5	order
ZS GPT-4	order platform
ZS Instruct	order platform
FS GPT-3.5	order platform
FS GPT-4	order platform
FS Instruct	order
RAG GPT-3.5	order platform
RAG GPT-4	order platform
RAG Instruct	order platform
FT Base	order platform
FT Instruct	order platform

TABLE VI
OUTPUT SAMPLE FOR NL2PROP DATASET

NLQ:	**"Show the orders opened with a lack of material at P-74"**
KQ (Ref):	**order situation Platform**
ZS GPT-3.5	order
ZS GPT-4	order situation Platform op
ZS Instruct	order platform op active
FS GPT-3.5	order situation 'technical object' Class Platform
FS GPT-4	order situation Platform
FS Instruct	order situation
RAG GPT-3.5	order situation Platform
RAG GPT-4	order situation Platform
RAG Instruct	order situation Platform
FT Base	order situation Platform
FT Instruct	order situation Platform

observe a significant difference in the performance obtained between both variants. Another practical option we found is the use of prompting techniques like RAG with GPT-4. This approach allows achieving similar performance to a fine-tuned model, but without the need to train adapters. It proves to be a versatile tool for all three types of datasets.

In general, although GPT-*n* models are impressive tools, their main disadvantage remains that they can not be executed

TABLE VII
OUTPUT SAMPLE FOR NL2CPV DATASET

NLQ:	**"Show the orders opened with a lack of material at P-74"**
KQ (Ref):	**order situation=open 'system status' contains MATF Platform=P-74**
ZS GPT-3.5	order, open, lack of material, P-74
ZS GPT-4	Order Situation contains 'OPEN' 'User Status' contains 'MATERIAL LACKING' Platform=P74
ZS Instruct	Order Situation='Open' Platform='P-74' "System Status"='Order Open'
FS GPT-3.5	order situation=open 'system status' contains material lack platform=P-74
FS GPT-4	order situation=open 'system status' contains 'material lack' Platform=P-74
FS Instruct	order situation=open 'system status' contains P-74 'user status' contains no material
RAG GPT-3.5	order situation=open 'system status' contains MATF Platform=P-74
RAG GPT-4	order situation=open 'system status' contains MATF Platform=P-74
RAG Instruct	order situation=open 'system status' contains MATF Platform=P-74
FT Base	order situation=open 'system status' contains MATF Platform=P-74
FT Instruct	order situation=open 'system status' contains MATF Platform=P-74

locally, which implies data uploading and data privacy risks. Another concern is the cost per query. This makes fine-tuning smaller models an interesting option if enough data and computational resources are available. It's crucial to consider various factors when selecting a tool for NLQs to KQs translation. The amount of data samples, computational resources, and data privacy all play a significant role. These considerations can guide future research and application of our findings.

For future work, we propose a deeper analysis of the generated queries to ensure the quality of the outputs by including human evaluators, implementing a human-in-the-loop approach, and adding other metrics, such as user satisfaction with the generated queries or the inclusion of GPT-*n* models within the fine-tuning technique.

ACKNOWLEDGMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and the National Council for Scientific and Technological Development – CNPq under project number 314121/2021-8.

REFERENCES

[1] A. H. Abdi, X. Tang, J. Eichelbaum, Das *et al.*, "Nl2kql: From natural language to kusto query," *arXiv preprint arXiv:2404.02933*, 2024.
[2] L. Lu, J. An, Y. Wang, C. Kong, Z. Liu, S. Wang, H. Lin, M. Fang, Y. Huang, E. Yang *et al.*, "From text to cql: Bridging natural language and corpus search engine," *arXiv preprint arXiv:2402.13740*, 2024.
[3] M. Staniek, R. Schumann, M. Züfle, and S. Riezler, "Text-to-overpassql: A natural language interface for complex geodata querying of openstreetmap," *arXiv preprint arXiv:2308.16060*, 2023.
[4] Z. Gu, J. Fan, N. Tang, S. Zhang, Y. Zhang, Z. Chen, L. Cao, G. Li, S. Madden, and X. Du, "Interleaving pre-trained language models and large language models for zero-shot nl2sql generation," *arXiv preprint arXiv:2306.08891*, 2023.
[5] Z. Gu, J. Fan, N. Tang, L. Cao, B. Jia, S. Madden, and X. Du, "Few-shot text-to-sql translation using structure and content prompt learning," *Proceedings of the ACM on Management of Data*, 2023.
[6] S. Chang and E. Fosler-Lussier, "Selective demonstrations for cross-domain text-to-sql," *arXiv preprint arXiv:2310.06302*, 2023.
[7] ——, "How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings," *arXiv preprint arXiv:2305.11853*, 2023.
[8] X. Liu, S. Pan, Q. Zhang, Y.-G. Jiang, and X. Huang, "Generating keyword queries for natural language queries to alleviate lexical chasm problem," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018.
[9] A. Kumar, S. Dandapat, and S. Chordia, "Translating web search queries into natural language questions," *arXiv preprint arXiv:2002.02631*, 2020.
[10] B. Kim, H. Choi, H. Yu, and Y. Ko, "Query reformulation for descriptive queries of jargon words using a knowledge graph based on a dictionary," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.
[11] H.-J. Song, A.-Y. Kim, and S.-B. Park, "Translation of natural language query into keyword query using a rnn encoder-decoder," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017.
[12] K. S. Kalyan, "A survey of gpt-3 family large language models including chatgpt and gpt-4," *Natural Language Processing Journal*, 2023.
[13] J. Lehmann, P. Gattogi, D. Bhandiwad, S. Ferré, and S. Vahdati, "Language models as controlled natural language semantic parsers for knowledge graph question answering," in *European Conference on Artificial Intelligence (ECAI)*, 2023.
[14] I. Amazon Web Services, "What are foundation models?" 2024, accessed on 01 05, 2024. [Online]. Available: https://aws.amazon.com/what-is/foundation-models/
[15] I. IBM, "What are llms?" 2024, accessed on 01 05, 2024. [Online]. Available: https://www.ibm.com/topics/large-language-models
[16] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, 2020.
[17] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.
[18] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, 2019.
[19] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, 2020.
[20] I. Amazon Web Services, "What is rag?" 2024, accessed on 01 05, 2024. [Online]. Available: https://aws.amazon.com/what-is/retrieval-augmented-generation
[21] Microsoft, "What is rag?" 2024, accessed on 01 05, 2024. [Online]. Available: https://learn.microsoft.com/en-us/ai/playbook/technology-guidance/generative-ai/working-with-llms/fine-tuning
[22] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International conference on machine learning*, 2019.
[23] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, 2024.
[24] G. M. Garcia, J. G. Jiménez, A. B. N. Júnior, Y. T. Izquierdo, M. Lemos *et al.*, "Database access control in a database keyword search tool," in *Anais do XXXVIII Simpósio Brasileiro de Bancos de Dados*, 2023.
[25] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, 2004.
[26] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the Second Workshop on Statistical Machine Translation*, 2005.
[27] R. Rei, C. Stewart, A. C. Farinha, and A. Lavie, "COMET: A neural framework for MT evaluation," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.
[28] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," in *International Conference on Learning Representations*, 2020.