

Machine Translation Models for Structured Query Construction from Natural Language Queries: A Case Study

Franklin Cardenoso Fernández* Wouter Caarls**

* *Electrical Engineering Department, Pontifícia Universidade do Rio de Janeiro (e-mail: franklin@aluno.puc-rio.br)*

** *Electrical Engineering Department, Pontifícia Universidade do Rio de Janeiro (e-mail: wouter@puc-rio.br)*

Abstract: This study presents a comparison of structured query construction methods from natural language queries for custom proprietary data. We implemented three machine translation methods: traditional machine learning, recurrent neural networks (RNN) with LSTM modules, and a popular large language model (LLM). We evaluate their performance across a synthetic dataset built with proprietary data through different experiments, aiming to identify the strengths and weaknesses of each model, providing insights into their behaviours and effectiveness in terms of practical implementations. The results of this comparative study can facilitate informed decision-making for researchers and practitioners in natural language processing. In this case study, LLMs had a somewhat higher accuracy than RNNs, but at a high additional computational cost.

Keywords: natural language processing, machine learning, sequence-to-sequence, large language models, machine translation

1. INTRODUCTION

It is well known that natural language processing (NLP) has gained significant advancements in recent years, particularly in the domain of machine translation (MT), with the introduction of neural machine translation (NMT) systems. These systems achieved reasonable translations with the application of recurrent neural networks (RNN) and the encoder-decoder architecture, Bahdanau et al. (2014); Sutskever et al. (2014). However, the introduction of the attention mechanism and the transformer architectures were responsible for the current revolution of NLP in different tasks with the rising of large language models (LLM), Gillioz et al. (2020).

Given that the core idea of MT systems is to facilitate the translation of text from one language to another, this idea can be extended to translating a sequence of texts or characters from a source structure to a target one. Deep learning architectures implement this through sequence-to-sequence models (*seq2seq*), which analyse and understand patterns and semantics between the input/output sequences and enhance their versatility in translation systems, Yousuf et al. (2021).

This general idea has been exploited for different applications, such as subroutines summarising from source code, LeClair et al. (2019), code generation from generic descriptions, Akinobu et al. (2022), and the currently popular models for coding such as Codellama, Copilot, among others, Zheng et al. (2023); or more specific tasks such as keywords extraction, Borisov et al. (2021); Issa et al. (2023). A particular application of MT is the construction

of keyword-based queries from natural language queries or vice-versa, Kumar et al. (2020); Zhong et al. (2017). This task is particularly useful when users interact with search engines or web applications requiring specific instruction formats to retrieve the target data.

However, building keyword-based queries from natural language instructions when the exact words are not present in the input sequences may be problematic, especially if specific formats and proprietary data domains are required to make the systems work properly. According to its complexity, this task can be addressed in different ways, from the use of traditional machine learning methods such as support vector machines (SVM) using numerical representations of the input sequences, or RNN models based on *seq2seq* architectures with long-short term memory (LSTM) modules, Joo et al. (2021); to more recent tools such as LLMs through prompting techniques, Chataut et al. (2024).

In this work, we present a comparative analysis of different approaches for this keyword-based query construction. Specifically, we compare a traditional SVM method with a 2D-LSTM *seq2seq* model and the popular LLM Mistral-7B. Our purpose is to evaluate the performance of these models across synthetically generated data with different structures of keyword-based query (KQ) and natural language query (NLQ) samples to extract information about their strengths and weaknesses. By performing this evaluation, we aim to analyse their behaviours and effectiveness in terms of utility, performance and computational resources in this task to provide insights to facilitate in-

formed decision-making for researchers and practitioners in natural language processing.

This paper is structured as follows: Section 2 briefly describes the related works, Section 3 presents the theoretical background that supports this research and the implementations, Section 4 describes the problem addressed and the experimental methodology we followed. Section 5 and Section 6 present the obtained results and their discussion respectively, and finally, in Section 7 we present the conclusions and propose new guidelines for future work.

2. RELATED WORKS

The task of translating natural language queries (NLQ) into keyword-based queries (KQ) is not recent, an earlier approach, such as the one presented by Orsi et al. (2011), addresses this task by combining ontologies and patterns with natural language queries to support the users while they interact with a medical platform in mobile devices.

On the other hand, Liu et al. (2018) addresses the problem of the lexical chasm between NLQs and web documents by proposing an RNN architecture with an encoder and two decoders, where the first decoder extracts the query terms from the source and the second decoder generates the KQs by applying attention modules. A similar approach is presented by Song et al. (2017), who implements an RNN encoder-decoder model to perform the translation task by adding an attention mechanism to deal with long sequences. Another interesting approach is one developed by Kim et al. (2021), who, instead of using RNN models, proposes a graph search architecture using a graph neural network to extract the relational properties between the keywords and the NLQ input. Among the most popular approaches for NLQ translation to structured queries, we find those that convert NLQs into structured query language (SQL) queries, for example, Sanyal et al. (2021) or Montgomery et al. (2020)

Moreover, contrary to most previous work, Kumar et al. (2020) discusses a method proposed to generate NLQs from KQs, highlighting its application in a search engine system. They analyse the situation to enhance the user experience by recommending NLQs constructed from search keywords.

A more recent study is presented by Chataut et al. (2024), which uses prompt engineering with public datasets for keyword extraction with three popular LLMs such as Llama2-7B, GPT-3.5 and Falcon-7B and then analyses the hallucination impact in these models as well as perform an interesting evaluation in terms of model complexity and demand for computational resources.

3. THEORETICAL BACKGROUND

3.1 Machine translation systems

As its own name describes, a translation system transforms a given text written in a source language into an equivalent text in a target language, the term "machine" states that this task is performed by an implemented system, Goutte (2009). The goal of these systems is to learn how to translate from the source to the target language, given

the nature of the task, this is usually performed with the machine learning framework.

3.2 Support Vector Machines (SVM)

SVMs are a family of supervised ML models used for classification and regression tasks. They aim to find the optimal hyperplane between the data points to discriminate and, consequently, classify them into different classes. This is done by maximising the distances between the hyperplane and the nearest data points (support vectors), Mammone et al. (2009). These models have proven to be excellent text classifiers for different topics, being used as interesting baselines to test other implementations, Joo et al. (2021).

3.3 2D-LSTM Seq2Seq Model

This architecture was presented by Bahar et al. (2018) as an alternative model for NMT tasks; by employing multi-dimensional long short-term memory (MDLSTM) modules, this key innovation allows essentially to re-read the input sentence with each new token in the output, which is based on the previously generated token.

Unlike traditional NMT models that treat input-output sequences as one-dimensional vectors, this model applies 2D-LSTM units to align source and target sequences in a 2D grid. Basically, maps the inputs' hidden states (concatenating both directions) in one dimension of the grid, while the ground truth or the previously generated token is mapped in the other dimension during the training and inference, respectively.

3.4 Mistral-7B

Mistral-7B is a transformer and foundation model that was released in September 2023 as a very promising model in its size category. It outperformed previous open-weights LLM models such as Llama 2 13B, Llama 1 34B, and CodeLlama 7B, among others, in different benchmarks. The key differences of Mistral rely on the inclusion of grouped-query attention (GQA) modules and sliding window attention (SWA), which results in faster inference and the capability to handle longer sequences, respectively, Jiang et al. (2023). Due to its versatility and high adaptation capability, this model has been adopted by many academic and industrial researchers.

4. EXPERIMENTAL METHODOLOGY

4.1 Problem and overview

We address the problem of structured query translation from natural language instructions in a search engine that works with proprietary data¹ with the objective of helping users with their interactions when retrieving information from a database.

The existing system allows the users to interact with a retrieval information system by using KQ as inputs and returning the related information in a tabular fashion,

¹ Due to confidentiality, the data will not be explicitly shown, but we will describe its functioning. The language of the data is Portuguese.

as described in Garcia et al. (2023). These keywords are divided into three categories: classes, properties, and values. So, if a user wants to perform a search, they need to introduce a list of keywords, and the system matches every keyword to return the occurrences from the database. Some examples of this type of user keyword query are presented below:

- order situation=open "system status" contains MAT Platform=P-74
- Object Platform situation Name = FPPLB

The inconvenience with this particular system is that if the user is not familiar enough with it, his searches will probably return nothing. On the other hand, some properties may have the same label for different classes; consequently, the class must be appended before every property. Moreover, every class can have multiple text labels, confusing novel users, specifically if they need to search for specific content in the system.

Therefore, we aimed to implement an external feature that could enhance the system's actual functioning by allowing users to introduce their queries in a natural language fashion and, using this information, translate the given query into a keyword-based instruction that could be used by the actual system directly, making their interaction with the actual platform easier as shown in the following example:

- **Input:** What is the status of the objects on the FPPLB platform?
- **Output:** Object situation Platform=FPPLB

Notice that our intention is not to modify the actual system for retrieving the information but to create an additional layer on top of the input search system to create KQs from the NLQs. So, to perform this implementation, we tested different approaches and present the comparison results in this work.

4.2 Synthetic dataset generation

Given that our main intention is translating NLQs to KQs by using NLP approaches, the first challenge addressed was the nonexistence of input/output samples. Therefore, a synthetic dataset was built before any implementation.

First, according to some suggestions collected from expert system users, we performed the following operations: first, we reduced the dimensionality of the queries used by selecting the most common search topics, and then we retrieved the related keywords (classes, properties, and values) to build a set of possible queries that a normal user could use in the platform. The reduced database is composed of 7 classes, 39 properties, and $\sim 2.7k$ values.

Then, a basic KQ dataset was built with input/output keyword-based samples and using additional suggestions from the expert users, who provided common structures from the system's actual functioning. Although both input and output are KQs, the labels were cleaned such they returned the expected information, even when different textual descriptions were used in the input.

Once the basic KQs were tested and validated, a GPT-4 model using few-shot prompting was used to create

the respective NLQs inputs. In addition, to increase the diversity of the dataset, within the prompt, the GPT-4 model was asked to generate the NLQs using different variations: verbosity, simplicity, technicality, and detail.

It is worth mentioning that the final dataset combines the basic dataset, composed of inputs/outputs samples in a keyword-based format, and the NLQ dataset, composed of inputs in natural language format and outputs with the keyword-based format. Table 1 summarises the number of samples per input type.

Table 1. Number of samples of the synthetic dataset

Input type	Train	Test
KQ	7270	903
NLQ	7886	992
All	15156	1895

4.3 Implemented systems

Once the dataset was built, we experimented with three approaches: an SVM-based, an RNN-based, and an LLM-based model.

SVM-based This approach uses a support vector machine (SVM) model at its core. Before feeding the input data into the model, we performed a basic preprocessing step by removing non-technical stopwords from the NLQs to reduce noise, lowercasing the sentences, and replacing some known synonyms with their corresponding label keyword. Then, based on the implementation presented by Mai (2022), we converted the inputs to a numerical representation by extracting the relevant features from the NLQs with TF-IDF; this is an NLP technique that weights the importance of the words in a corpus text based on their occurrences, Baeza-Yates et al. (1999). Finally, the resulting sparse matrix was used to fit the SVM model. The model was fit with a *linear* kernel, a regularization parameter of 2, and we set the remaining hyperparameters to their default values in the `Sklearn` package.

RNN-based For this system, we based our implementation on the architecture presented by Bahar et al. (2018), a *seq2seq* model using 2DLSTM modules. To create the vocabulary set for the model, we used simple string split tokenization and the sequences were projected into a 128-dimensional embedding. The model was trained from scratch with an Adam optimizer and used a cross-entropy loss function to calculate the loss value between the predictions and the targets. We performed two kinds of training, one powered with a GPU card and the other using only the CPU processor. Table 2 summarises the hyperparameters we used for this implementation.

Table 2. Seq2seq model parameters

Parameter	Value
Batch-size	32
Epochs	50
Learning rate	0.0005
Embedding dimension	128
QLoRA Encoder dimension	64
2D-LSTM dimension	128
Dropout	0.2

LLM-based Given that foundational LLM models require a vast amount of data, starting a training process from scratch is almost impossible in these cases because of the data and computation requirements. Therefore, common techniques for using LLMs are prompting techniques or fine-tuning if the task is too specific. Given the nature of our task, we employed this last method. Specifically, we used a quantized low-rank adaptation (QLoRA) approach, Dettmers et al. (2024), which makes fine-tuning LLM models with lower computational resources possible. Through this method, the model’s parameters are frozen, and only attached adapters are trained, considerably reducing the number of trainable parameters and saving memory and calculations. For our purposes, we applied QLoRA to all the linear layers of the 4bit quantized Mistral-7B model. Furthermore, we used the following QLoRA parameters: the rank is 8, the scaling factor (alpha) is 16, and the dropout probability is 0.1.

4.4 Evaluation metrics

For all the experiments, we used different automatic metrics to evaluate the quality of the translations. We used widely known traditional metrics such as ROUGE-L, Lin (2004), which assesses the longest common subsequence; BLEU, Papineni et al. (2002), which measures n-gram precision; METEOR, Banerjee and Lavie (2005), which accounts for synonymy; and Exact Match, which checks for exact correspondence between generated and reference; as well as LLM-based metrics such as the well-known BertScore to calculate semantic similarity between the generated text and the reference, Zhang et al. (2020).

4.5 Experimental setup

All the implementations were done in a desktop workstation with a 20-core Intel® Core™ i7-12700 CPU as the processor, 64 GB of RAM, a GeForce RTX 3090 graphic card, and Ubuntu 20.04.6 LTS as the operating system. Additionally, for all the models, Python was used as the programming language with the following packages for the training process: `Sklearn` for the SVM, `Pytorch` for the RNN and the LLM, and for the evaluation process, we used the metrics of the `evaluate` library from the Huggingface implementation for all the models.

5. RESULTS

Several experiments were performed to evaluate the effectiveness of the implemented systems. First of all, we summarise the accuracy with the metric values obtained for the complete testing dataset (Table 3). Furthermore, we present Table 4 and Figure 1, which summarises the time for training and testing required by the models to complete all the operations and the amount of memory required. Notice that we present the results for additional tests by using only CPU for training and testing in the RNN model, while for the LLM model, CPU is only used to perform inference; this was done intentionally because, with the actual libraries, quantisation is only possible in GPU for training. An additional consideration is that although the testing time for LLM-based or RNN-based could be lower when passed a batch of samples, we are

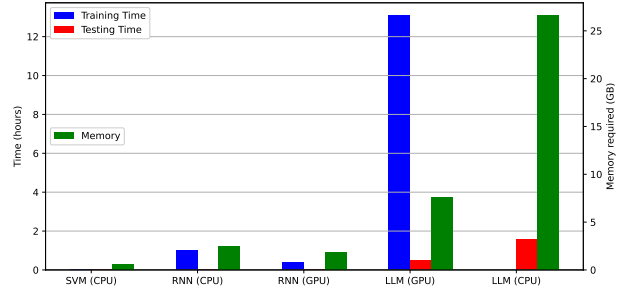


Fig. 1. Time/memory usage per model with the complete dataset.

performing the inference separately for every sample, as that is how it would be deployed.

Table 3. Metric results for the complete testing dataset

System	SVM	RNN	LLM
ROUGE-L	0.850	0.923	0.968
Exact Match	0.328	0.765	0.812
BLEU	0.627	0.838	0.91
METEOR	0.826	0.913	0.946
BertScore (P)	0.940	0.976	0.988
BertScore (R)	0.929	0.973	0.987
BertScore (F1)	0.934	0.975	0.988
Average	0.776	0.91	0.94

Table 4. Time (in seconds) and memory (in GB) required by the models

System	Training time	Testing time	GPU used	Memory
SVM(CPU)	22	47	-	~ 0.6
RNN(CPU)	3678	34	-	~ 2.5
RNN(GPU)	1399	36	✓	~ 1.8
LLM(GPU)	47113	1755	✓	~ 7.6
LLM(CPU)	-	5685	-	~ 26.6

6. DISCUSSION

We can extract different insights from the results presented in the previous section. First of all, Table 3 shows that the RNN-based and the LLM-based systems perform similarly (above 90% avg. acc. approx.) while the SVM-based system presents an average lower accuracy (around 77%). However, the BertScore metric results show that the generated KQs are very similar to the labels, which could indicate that the SVM-based model can capture most of the KQ structures.

To perform a deeper analysis, we split the output metrics according to the input type in the complete dataset. This evaluation is presented in Table 5 and Table 6. These results show that the RNN and LLM models perform better when dealing with KQs and decrease when dealing with NLQs, but surprisingly, SVM performs better with NLQs as inputs. In addition, it can be seen that while both the RNN and LLM models perform perfectly for the KQs as inputs, the performance for the RNN model decreases more for the NLQ inputs compared to the LLM model (Fig. 2).

However, the largest differences we could find come from the practical implementation of the systems (Table 4 and Fig. 1). For example, while the LLM-based system

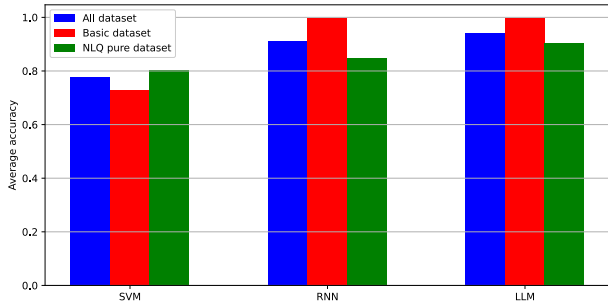


Fig. 2. Average accuracy metrics for the models

clearly outperforms the remaining implementations, its computational requirements are also higher, especially in the training stage, where the necessity of a GPU is practically mandatory since fine-tuning LLM in CPU is too time-consuming, and it is not recommended practice. First, the LLM(GPU) model takes around 13 hours to be trained, while it only takes around 1 second per sample to perform the inference (if we use CPU for inference, the model is 3x slower).

On the other hand, although the RNN-based system requires around 1 hour (CPU) or 23 min (GPU) to be trained, the inference time is reduced drastically to 18.46 ms per sample approximately for both engines: CPU or GPU, surprisingly, outperforming the inference time required by the SVM-based model (25% lower). This could be probably due to the RNN-based model being powered by a GPU card, which accelerates the calculations, while the SVM model only uses a CPU, or probably because the library used for the RNN implementation optimizes the resources better.

For this particular study case, the RNN-based model presents a better trade-off between accuracy and computational requirements than the remaining implementations, given that it has low memory consumption, a lower inference time, and reasonable accuracy. This allows us to perform another kind of comparison. For example, using the RNN(GPU) implementation as our baseline, we can discover how the other models perform against the base-

Table 5. Metric results for the KQ inputs

System	SVM	RNN	LLM
ROUGE-L	0.767	1	1
Exact Match	0.329	1	1
BLEU	0.483	1	1
METEOR	0.729	0.976	0.976
BertScore (P)	0.938	1	1
BertScore (R)	0.926	1	1
BertScore (F1)	0.932	1	1
Average	0.729	0.997	0.997

Table 6. Metric results for the NLQ inputs

System	SVM	RNN	LLM
ROUGE-L	0.925	0.863	0.939
Exact Match	0.326	0.552	0.641
BLEU	0.644	0.813	0.896
METEOR	0.913	0.856	0.919
BertScore (P)	0.941	0.954	0.978
BertScore (R)	0.931	0.949	0.977
BertScore (F1)	0.936	0.952	0.977
Average	0.802	0.848	0.904

line using different criteria. For instance, the LLM(GPU) model takes around $48.75\times$ more time for inference, requires $4.2\times$ more memory, and is only $1.03\times$ more accurate. On the other hand, the SVM(CPU) solution has a lower memory consumption and is $0.85\times$ less accurate, but its training is $0.016\times$ faster. These kinds of insights can be inferred from Table 7.

However, it is important to note that besides these criteria, the final implementation choice will be conditioned by extra variables, such as computational resource availability, data complexity, and outcome preferences, and whether more accurate or faster responses are targeted. For example, we must notice that although the training step can demand larger computational resources, this process is made only once, while inference is made every time the system is called to translate a query. Consequently, we could focus more on inference, memory usage, accuracy criteria or scalability.

Table 7. Comparison against RNN (GPU)

System	Training time	Inference time	Memory usage	Avg. Acc.
SVM(CPU)	$0.016\times$	$1.3\times$	$0.33\times$	$0.85\times$
RNN(CPU)	$2.6\times$	$0.94\times$	$1.39\times$	$1\times$
RNN(GPU)	$1\times$	$1\times$	$1\times$	$1\times$
LLM(CPU)	-	$158\times$	$14.7\times$	$1.03\times$
LLM(GPU)	$33.6\times$	$48.75\times$	$4.2\times$	$1.03\times$

7. CONCLUSION AND FUTURE WORK

During this work, we described a comparative analysis of three approaches to constructing structured queries from natural language queries. One is based on traditional ML with an SVM model at its core, the other on an RNN model with 2DLSTM modules, and finally, a third implementation based on the nowadays popular LLMs, Mistral-7B for our case, was implemented. For our particular study case, our experiments have shown that although recent developments achieve good results, even a traditional method can generate reasonable metric values.

Our findings allow us to say that if we look for a better trade-off between accuracy and computational resources, the RNN method is a remarkable choice. On the other hand, if we are looking for more accurate results, LLMs appear to be an obvious selection. Finally, traditional ML models are still an interesting option if we lack computational resources and only intend to achieve a reasonable result. However, besides these conditions, this election will also be conditioned by other variables, such as dataset or task complexity. In summary, our final choice will function according to the conditions we described and, as stated before, will be conditioned by what we seek.

In future work, we would like to including a qualitative measurement for comparison, being the user satisfaction with the generated results. This would involve carefully observing and analysing the generated KQs to discover other characteristics of the assessed models, such as hallucinations.

ACKNOWLEDGEMENTS

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

- Akinobu, Y., Kajiura, T., Obara, M., and Kuramitsu, K. (2022). Nmt-based code generation for coding assistance with natural language. *Journal of Information Processing*.
- Baeza-Yates, R., Ribeiro-Neto, B., et al. (1999). *Modern information retrieval*. ACM press New York.
- Bahar, P., Brix, C., and Ney, H. (2018). Towards two-dimensional sequence to sequence model in neural machine translation. *arXiv preprint arXiv:1810.03975*.
- Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- Banerjee, S. and Lavie, A. (2005). Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*.
- Borisov, O., Aliannejadi, M., and Crestani, F. (2021). Keyword extraction for improved document retrieval in conversational search. *arXiv preprint arXiv:2109.05979*.
- Chataut, S., Do, T., Gurung, B.D.S., Aryal, S., Khanal, A., Lushbough, C., and Gnimpieba, E. (2024). Comparative study of domain driven terms extraction using large language models. *arXiv preprint arXiv:2404.02330*.
- Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2024). Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*.
- Garcia, G.M., Jiménez, J.G., Júnior, A.B.N., Izquierdo, Y.T., Lemos, M., Casanova, M.A., Ferreira, A.C., and da Silva, F.P.T. (2023). Database access control in a database keyword search tool. In *Anais do XXXVIII Simpósio Brasileiro de Bancos de Dados*. SBC.
- Gillioz, A., Casas, J., Mugellini, E., and Abou Khaled, O. (2020). Overview of the transformer-based models for nlp tasks. In *2020 15th Conference on Computer Science and Information Systems*. IEEE.
- Goutte, C. (2009). *Learning machine translation*. MIT Press.
- Issa, B., Jasser, M.B., Chua, H.N., and Hamzah, M. (2023). A comparative study on embedding models for keyword extraction using keybert method. In *2023 IEEE 13th International Conference on System Engineering and Technology*.
- Jiang, A.Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D.S., Casas, D.d.l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. (2023). Mistral 7b. *arXiv preprint arXiv:2310.06825*.
- Joo, H., Burns, M., Kalidaikurichi Lakshmanan, S.S., Hu, Y., and Vydiswaran, V.V. (2021). Neural machine translation-based automated current procedural terminology classification system using procedure text: Development and validation study. *JMIR formative research*.
- Kim, B., Choi, H., Yu, H., and Ko, Y. (2021). 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*.
- Kumar, A., Dandapat, S., and Chordia, S. (2020). Translating web search queries into natural language questions. *arXiv preprint arXiv:2002.02631*.
- LeClair, A., Jiang, S., and McMillan, C. (2019). A neural model for generating natural language summaries of program subroutines. In *2019 IEEE/ACM 41st International Conference on Software Engineering*. IEEE.
- Lin, C.Y. (2004). ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics.
- Liu, X., Pan, S., Zhang, Q., Jiang, Y.G., and Huang, X. (2018). 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*.
- Mai, F. (2022). Tf-idf + svm baseline. URL <https://www.kaggle.com/code/hsrobo/tf-idf-svm-baseline/script>. Accessed on 01 01, 2024.
- Mammone, A., Turchi, M., and Cristianini, N. (2009). Support vector machines. *Wiley Interdisciplinary Reviews: Computational Statistics*.
- Montgomery, C., Isah, H., and Zulkernine, F. (2020). Towards a natural language query processing system. In *2020 1st International Conference on Big Data Analytics and Practices*. IEEE.
- Orsi, G., Tanca, L., and Zimeo, E. (2011). Keyword-based, context-aware selection of natural language query patterns. In *Proceedings of the 14th International Conference on Extending Database Technology*.
- Papineni, K., Roukos, S., Ward, T., and Zhu, W.J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*.
- Sanyal, H., Shukla, S., and Agrawal, R. (2021). Natural language processing technique for generation of sql queries dynamically. In *2021 6th International Conference for Convergence in Technology (I2CT)*. IEEE.
- Song, H.J., Kim, A.Y., and Park, S.B. (2017). 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*.
- Sutskever, I., Vinyals, O., and Le, Q.V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*.
- Yousuf, H., Lahzi, M., Salloum, S.A., and Shaalan, K. (2021). A systematic review on sequence-to-sequence learning with neural network and its models. *International Journal of Electrical & Computer Engineering*.
- Zhang, T., Kishore, V., Wu, F., Weinberger, K.Q., and Artzi, Y. (2020). Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.
- Zheng, Z., Ning, K., Wang, Y., Zhang, J., Zheng, D., Ye, M., and Chen, J. (2023). A survey of large language models for code: Evolution, benchmarking, and future trends. *arXiv preprint arXiv:2311.10372*.
- Zhong, V., Xiong, C., and Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.