VIETNAMESE TEXT-TO-SQL WITH LARGE LANGUAGE MODELS: A COMPREHENSIVE APPROACH

Anonymous authors
Paper under double-blind review

ABSTRACT

In the current era of Artificial Intelligence (AI), the realm of database querying is experiencing a profound evolution. With the recent emergence of Large Language Models (LLMs), with a particular emphasis on Vietnamese in this study, a promising opportunity arises to bridge the gap between human language and database interactions. In this paper, we embark on realizing this vision through a three-pronged approach. Firstly, we introduce a few-shot learning method designed to enhance the database schema comprehension of Vietnamese LLMs. Secondly, we employ a chain-of-thought technique to systematically guide LLMs in capturing complex natural language expressions for SQL generation. Thirdly, we introduce a novel method to streamline the input schema by removing redundant parts and retaining only the parts that are truly relevant to enhance the efficiency and accuracy of the SQL generation process. Finally, we experimented with a combination of few-shot, chain-of-thought learning, and schema-enhancing methods. Through experimentation with augmented datasets, we observe encouraging initial results. Our approach outperforms the current state-of-the-art model by 23% in exact matching on the Vietnamese ViText2SQL dataset. We achieved this result with a single pre-training step and one epoch of retraining, compared to the SoTA model's 10 epochs. These findings demonstrate the effectiveness of our method and its potential for Vietnamese text-to-SQL applications.

Keywords: *Text-to-SQL, Large Language Models, Few-shot, Chain-of-thought, Mini Schema, ViText2SQL, SQL*

1 INTRODUCTION

In the current era of Artificial Intelligence (AI), the realm of database querying is experiencing a profound evolution. Globally, text-to-sql research started quite early. In previous studies, there are a lot of methods, ranged from filling values into query templates to training seq2seq encoder-decoder models, evaluating them on English text-to-sql datasets like SPIDER Yu et al. (2018). In Vietnam, there is a lack of research in the text-to-sql task. An initial study by VinResearch Nguyen et al. (2020) published a Vietnamese text-to-sql dataset translated from the SPIDER dataset Yu et al. (2018) with two levels: syllable (e.g., hom nay toi di hoc) and word (e.g., hom_nay toi di hoc), and benchmarked it with two selected seq2seq-based models, EditSQL Zhang et al. (2019) and IRNet Guo et al. (2019), for the task of directly generating SQL on this dataset.

In this study, we fine-tuned the Large Language Models on specific text-to-sql tasks using various methods, such as generating SQL queries directly from questions and inferring SQL components step-by-step before responding with Chain-of-Thought learning. Additionally, we employed Few-Shot learning to adapt the model to similar contexts. This approach is expected to generate SQL queries with higher accuracy than the current state-of-the-art models, thereby increasing trust in the results. Finally, we introduced a novel method to streamline the input schema by removing redundant parts. This method is expected to reduce noise in the schema, thereby increasing the accuracy of the SQL generation process.

Through experiments, the model trained with the few-shot method outperformed by 23% (meaning 79.4%) exact matching metric on the ViText2SQL test dataset Nguyen et al. (2020), after being pretrained once on our dataset and retrained for 1 epoch on ViText2SQL, compared to the current state-of-the-art model trained for 10 epochs with 52.8% on the syllable level.

Moreover, we modify two metrics, component matching and execute matching, for Vietnamese query evaluation from the SPIDER evaluator. Our models also outperform on component matching for each query components (SELECT, WHERE, ORDER BY, GROUP BY, and KEYWORDS) and achieves the highest execution accuracy, with 88.2% at the syllable level and 87.3% at the word level.

To summarize, in this study, we contributed:

- Experiments with few-shot, chain-of-thought (CoT) training methods, and variant approaches like FewShot CoT combinations.
- New schema filtering method, by training a retriever model specifically designed for filtering input schema.
- Modifications to the evaluator to assess the performance of Vietnamese queries.

2 Related Works

In the current era of Artificial Intelligence (AI), the realm of database querying is experiencing a profound evolution. Globally, text-to-sql research started quite early. Studies Seq2Sql Zhong et al. (2017), SParC Yu et al. (2019) and RASAT Qi et al. (2022) have approached methods such as filling values into SQL templates according to grammar, combining database schema and questions to enhance context, and integrating graphs. Recent state-of-the-art models on the Spider dataset, like RYANSQL Choi et al. (2020) and RAT-SQL Wang et al. (2021), use seq2seq encoder-decoder architectures.

For published datasets, SPIDER Yu et al. (2018) is one of the prominent ones, comprising 10,181 questions and answers along with 5,693 SQL queries across 200 databases. However, the shared dataset contains 9,691 question-answer pairs, 5,263 SQL queries, and 166 databases. Additionally, it proposes a theoretical framework for evaluating query complexity by levels of Easy, Medium, Hard, and Very Hard. It also includes three metrics for assessing query performance: Exact Matching, Component Matching, and Execute Matching.

In recent years, Large Language Models (LLM) have demonstrated their potential in solving SQL problems in natural language. There has been a lot of research on this topic and demonstrated the applicability and effectiveness of these methods. In particular, Gao et al. (2023) research has conducted a comprehensive evaluation of the ability of prompt engineering methods, discussed benchmark data sets and fine-tuning methods in solving text-to-SQL problems.

In Vietnam, a study at VinResearch Nguyen et al. (2020) shows that two learning techniques EditSQL and IRNet, achieve higher accuracy and determine two approaches in the Vietnamese context, which is a monosyllabic language. Therefore, the research team investigates two levels: syllable (e.g., hom nay toi di hoc) and word (e.g., hom_nay toi di hoc). Moreover, the research also publishes a Vietnamese text-to-sql dataset translated from the SPIDER dataset, called ViText2SQL dataset [1].

3 Background

3.1 Instruction Tuning

Instruction Tuning is a training method for pre-trained language models for a specific task. The data consists of two components: Instructions and Outputs (or Targets). Instructions contains user requirements, input information, task descriptions, and user expectations that the language model must fulfill. The output is the string the model is required to generate. Instruction tuning is an important technique to help pre-trained language models quickly adapt to real-world tasks, while controlling the ability to generate results and achieve high performance.

[1] https://github.com/VinAIResearch/ViText2SQL

3.2 Low-Rank Adaptation Training

LoRA (Low-Rank Adaptation) is a method for fine-tuning pre-trained models that was introduced by Hu et al. (2021). It has several advantages over traditional fine-tuning methods: LoRA requires calculating gradients for only a small number of new parameters while keeping the original model fixed, making it more efficient than fine-tuning the entire model. The LoRA weight set is typically only 1-2% the size of the original weights, allowing easier storage and deployment. Multiple LoRA weight sets can be applied to the same original model since the original weights remain unchanged, letting a model be adapted to multiple tasks. LoRA can also be combined with other training methods to further boost performance for different tasks. LoRA weights merge with the original model during deployment, avoiding additional inference latency. In principle, LoRA can be applied to any subset of weight matrices in a neural network to reduce the number of trainable parameters. However, typically in Transformer models LoRA is applied to attention blocks only. The resulting number of trainable parameters in a LoRA model depends on the size of the low-rank update matrices, which is determined mainly by the rank r.

Applying LoRA to Large Language Models with billions of parameters may not fit in memory. Tim Dettmers and Artidoro Pagnoni Dettmers et al. (2023) proposed quantizing pretrained models before using LoRA fine-tuning. Quantization LoRA (QLoRA) uses NF4 to quantize original weights to 4-bit, drastically reducing size. LoRA then adds parameters to the quantized model.

3.3 Prompt Engineering Techinque

There are several prompt techniques in training Language Model Models (LLMs). In this study, we chose Chain-of-Thought and Few-Shot learning to investigate the model behavior on the text-to-sql task due to the reasons outlined above.

3.3.1 Chain-of-Thought

Chain-of-Though Learning Prompting Liu & Tan (2023) is a new prompting technique designed to get large language models to explain their reasoning. Unlike traditional prompting which just seeks an answer, CoT prompting requires the model to provide an answer and explain the steps taken. This approach has proven effective in improving results for tasks like arithmetic, common sense reasoning and symbolic reasoning. By encouraging a more detailed reasoning process, CoT prompting greatly enhances how we interact with LLMs. This leads to more accurate and interpretable outputs, especially for complex reasoning tasks. CoT prompting works particularly well with larger models, highlighting potential to develop AI that gives right answers along with transparent thought insights. This helps connect human and artificial reasoning.

Designing effective CoT prompts requires understanding the task and reasoning steps involved. It is crucial to meticulously design the reasoning process and clearly document each step involved. Additionally, the model can be guided to generate multiple inferences, collectively termed CoT consistency, and employ a voting mechanism to determine the final outcome.

3.3.2 Few-shot Learning

Few-shot learning involves training machine learning models with a small amount of data to guide predictions, unlike standard techniques using large training data. In Natural Language Processing (NLP), it can be used with large language models pre-trained on large text datasets. This enables the model to understand related unseen tasks from just a few examples by learning tasks implicitly.

Few-shot examples contain: a short task description; a few input-output examples showing expectations; and a prompt example for the model to complete. Crafting these is tricky as models are sensitive to example wording. A way to optimize it is learning a shared representation across tasks then training task-specific classifiers on top of this.

Note that the number of few-shot examples can impact effectiveness. Too few might not provide enough guidance, while too many might limit generalization.

3.3.3 Few-shot and Chain-of-Thought Combine Learning

Combining few-shot learning and Chain-of-Thought prompting is a method being researched to enhance large language models' ability to learn complex tasks with limited data. Chain-of-Thought prompts can guide the model's reasoning with few examples, making learning more effective. It explains the model's thought process, allowing understanding of how it answers and identifying biases to build trust. Prompts help break down complex tasks into simpler steps, strengthening the model's capacity for new concepts with restricted data.

3.4 Retrieval Bi-Encoder

Retrieval Bi-encoder is a system using dense retrieval with a bi-encoder for retrieving documents/information that are relevant for the search query. In a bi-encoder model, there are two seperate encoders - one for encoding the input query, and another for encoding the candidate documents/information. The goal of bi-encoder is to try to map the meaning between two inputs together. Therefore, Retrieval Bi-Encoders are commonly used in document retrieval, such as search engines.

3.5 CodeLLama - A Large Language Model for Code

In August 2023, Meta AI unveiled the foundation model CodeLlama Rozière et al. (2024), inheriting the parameter set of Llama2 and further trained on a programming language dataset comprising over 500 billion tokens. Subsequently, the model was fine-tuned with an instruction-based approach using 5 billion tokens for user interaction. Consequently, the CodeLlama-Instruct model handles code generation tasks based on specific descriptions, including filling in missing code segments. CodeLlama offers three different sizes: (7B, 13B, 34B), each with variants such as CLM, Python code generation, and instruction-based response. CodeLlama supports multilingualism, including Vietnamese.

In this study, CodeLlama was selected as the baseline model for the text-to-sql task due to its robust Natural Language Processing (NLP) capabilities and superior performance in domains related to code. CodeLlama's strong NLP capabilities enable it to effectively comprehend and analyze complex SQL queries, facilitating a seamless conversion from text to SQL compared to other models. Moreover, CodeLlama has consistently demonstrated high performance in various NLP tasks, including text-to-sql. Benchmark evaluations across diverse code benchmark datasets have shown that CodeLlama outperforms starCoder Li et al. (2023) and Qwen Bai et al. (2023), further solidifying its position as a suitable baseline model for text-to-sql tasks.

4 Dataset Construction

4.1 Pre-trained Dataset

This dataset is a super-dataset related to text-to-sql task with general domain and general type query includes 3 datasets:

- Text2SQL: Comprising 240,000 synthesized and translated data from other Text2SQL datasets worldwide, using Google Translate API and Gemini API 4.3 to translate to Vietnamese. It includes questions, database schemas, and SQL queries.
- Few-Shot: Consisting of 240,000 Few-Shot data from the Text2SQL dataset. The detail steps to construct few-shot showed at 4.4.
- Chain-of-Thought: Comprising 60,000 filtered data from the Text2SQL dataset, summarizing the step-by-step reasoning process using the Gemini API.

4.2 Text-to-SQL Dataset Construction

Currently, the only publicly available Vietnamese text-to-sql (text2sql) dataset is ViText2SQL Nguyen et al. (2020). However, this dataset is relatively small (approximately 10,000 samples) and may not be suitable for training large language models (LLMs) due to the risk of overfitting. To address this limitation, we propose a new Vietnamese text-to-sql dataset that is both generalizable

across various domains (e.g., healthcare, education, sports, economics) and diverse in terms of query length and complexity. This dataset aims to serve as a valuable pretraining resource for LLMs to effectively perform text-to-sql tasks of varying difficulty. The construction of the dataset involved the following steps:

- English Text-to-SQL Datasets Collection: A collection of approximately 240,000 English samples was gathered from 13 reputable open-source sources, including Spider, WikiSQL, sql_create_context, and Squall. These datasets ensure diverse query structures, encompassing operations such as FROM, WHERE, GROUP BY, ORDER BY, AGG, JOIN, and INTERSECT.
- Machine Translation using Google Translate API: The English datasets were translated into Vietnamese using the Google Translate API. To facilitate efficient translation and minimize API usage, a keyword detection approach was employed. Keywords were replaced with special tokens ##, allowing for subsequent translation of the remaining Vietnamese text and tokenization.
- Postprocessing: The translated data underwent post-processing to address translation errors, such as extra whitespace, improper placement of Vietnamese field names in quotation marks, and inconsistencies in double and single quotes.
- Tokenization: Once the data was cleaned, the queries and schemas were tokenized at the syllable level. To introduce word-level representations, the word_tokenize function from the UnderTheSea library was applied.

4.3 Chain-of-Thought Dataset Construction with Gemini API Prompt Engineering

To generate text-to-sql data from the Gemini API, besides receiving the SQL query, the response also includes unnecessary expressions. Therefore, to extract the correct information from the response, our team has specified in the prompt that the result must be enclosed within opening and closing tags. For example, the SQL query needs to be enclosed within the opening tag `<sql>` and the closing tag `</sql>`. Similarly, the inference Chain-of-Thought process must be enclosed within the opening tag `<CoT>` and the closing tag `</CoT>`. We defined 3 kinds of Chain-of-Thought reasonings:

- Components Inference: is the process of simple inference, filling in the information from the question into a basic SQL template appropriately.
- Query Type Inference: is a more advanced process where initially, the model must predict the type of query (e.g., JOIN IN, EXCEPT, NESTED query, etc.), and then complete each part with information from the question and synthesize it into an SQL query.
- Schema-based Inference: is the process where the model selects the most relevant tables from the question among many tables in the database schema. Then it selects suitable columns (fields) from the most relevant tables based on the information from the question. It synthesizes the information to generate an SQL query.

To verify the correctness of CoT generation, our team employs Gemini to summarize the information from the CoT and generate a complete SQL query. We then compare the generated SQL query with the ground truth. If they match, we conclude that the CoT generation is correct; otherwise, we conclude that it is incorrect.

4.4 Few-Shot Dataset Construction with K-means Algorithm and PhoBERT Embedding

To achieve effective few-shot learning, the quality of the training data is crucial. This paper presents a methodology for constructing a high-quality few-shot dataset for Vietnamese NLP tasks. The dataset is designed to ensure high similarity between shots and target samples in terms of both query structure and text content. Additionally, the dataset incorporates diversity in domains and sentence lengths to enhance the generalization ability of the models. The dataset construction process involves the following steps:

- Keyword List Creation: A comprehensive list of SQL keywords is created, ranging from common keywords like SELECT, FROM, WHERE, and AGG to less frequent ones like INTERSECT and EMPTY.
- TF-IDF Vector Embeddings: TF-IDF (Term Frequency-Inverse Document Frequency) is employed to generate vector embeddings for SQL query sentences using the keyword list.
- K-means Clustering: K-means clustering is applied to group similar SQL query sentences based on their TF-IDF vector embeddings.
- Cluster Selection: The Elbow method and silhouette score are utilized to determine the optimal number of clusters. In this case, 12 clusters are chosen. After clustering, the properties of each cluster are examined. For instance, cluster 0 contains queries with the pattern "SELECT ... FROM ... WHERE ...", while cluster 8 consists of queries with the pattern "SELECT AGG(...) FROM ... WHERE ...".
- PhoBERT Embeddings: For each pair of query and text within a cluster, PhoBERT, a Vietnamese language model, is used to generate text embeddings. The goal is to group semantically similar question text together using similarity search based on embeddings.
- Similarity Score Calculation: Within each cluster, 200 random samples are selected, consisting of a query-text pair and a target sample. Similarity scores are computed between the embeddings of the samples and the target sample. This approach reduces computational burden and execution time while ensuring diversity within shots and preserving similarity.
- Shot Selection: The five shots with the highest similarity scores are chosen for each sample.

In few-shot learning, it is crucial to maintain fairness during training and testing to evaluate the model's true performance. To achieve this, a fundamental principle must be adhered to: no shot constructed from the training set should contain elements from the test set. Conversely, the test set should exclusively comprise shots derived from the training set. This segregation ensures that the model is not exposed to test data during training, preventing overfitting and providing a more accurate assessment of its generalization ability.

4.5 Schema filtering Dataset Construction with GPT-4o API

To create a Retriever for schema filtering, we take the questions, queries and schemas from our pretrained dataset and use GPT-4o to extract the tables in the schema that are actually relevant to the question and query, discarding the rest. We designed the prompt effectively so that the new schema is answered by a special tag #mini-schema.

To verify the correctness of schema filtering, we determined the similarity between "Schema" (original schema) and "MiniSchema" (filtered schema) using Term Frequency-Inverse Document Frequency (TF-IDF). This allowed us to confirm that the schema filtering was accurate. The "MiniSchema" typically has fewer SQL statements than the "Schema." A straightforward comparison of similarity would yield a relatively low score, which is inaccurate in reflecting their relationship. So, we first split the SQL statements in both "Schema" and "MiniSchema" based on the semicolon. Then, for each SQL statement in the "MiniSchema", we calculate its similarity with each statement in the "schema." We select the maximum similarity score for each comparison, forming a list of these maximum values. Finally, the similarity score of the schema and mini schema is calculated by taking the average of the list of maximum similarity scores. After testing, a similarity score greater than 0.75 ensures that SQL statements in mini schema also appear in "Schema".

4.6 ViText2SQL Dataset Construction with Chain-of-Thought and Few-Shot

In addition to the original few-shot and chain-of-thought (CoT) evaluation tasks defined in the pretrained dataset, we introduce similar evaluation setups within the ViText2SQL Nguyen et al. (2020) benchmark dataset. This extension allows for a more comprehensive assessment of model performance across diverse text-to-sql scenarios.

5 PROPOSED APPROACH

In this study, we explore the selection and fine-tuning of a baseline model for code generation. We use CodeLLama for experiment. Due to resource constraints and training time and dataset limitations, we opted for a fine-tuning approach using Qlora rather than full-parameter training of the base model. To identify the optimal lora-rank, we conducted experiments and evaluated based on training loss, validation loss, and metrics measuring the alignment between model output and ground truth.

After selecting the best rank, we used the two versions of pre-trained dataset to perform fine-tuning on the general text-to-sql task: full schema version and mini-schema version. For full schema version, this a the standard pre-trained dataset that we have prepared by synthesizing from many sources and using Gemini API to translate into Vietnamese. For mini-schema version, we have:

- Training phase: We have extracted the mini schema version from the original schema using the GPT-4o API.
- Testing phase: We trained a bi-encoder retriever using a pre-trained mini-schema dataset. The goal is that during inferencing, the schema of each sample will be passed through this retriever layer first to filter out only the tables relevant to the user's question. This approach helps to reduce the amount of redundant information, noise and increases the accuracy of SQL query generation.

Traditionally, retriever models are trained with natural language data. However, for this problem, to build a retriever which is robust in linking natural question input and SQL code output, we implemented a three-stage training process. Each stage is designed with increasing difficulty, helping the model gradually adapt and better recognize the correlation between text and SQL code:

- We chose BGE-M3[2] as our embedding model because it is a multi-linguality model that has been trained with Vietnamese data and has achieved quite good benchmark results on the embedding model rankings[3]. To enhance the ability to represent Vietnamese language, we continue to train this model on the Masked Language Model (MLM) task with a Vietnamese dataset, combining Vietnamese data (general topics) and SQL code, with about 500k samples, training in 1 epoch.
- Phase 1 - Query to Translated query: User queries were translated into Vietnamese using the GPT-4o API, then it will be mapped into set of sentence pairs. The goal is to help the model initially learn the similarity between Vietnamese language and SQL code. Since we have set of sentence pairs between SQL query and translated SQL query, which are also called positive pairs, the *Multiple Negatives Ranking loss* (MNR loss) is a great loss function to use.
- Phase 2 - Translated query to Mini-schema: Instead of making translated SQL queries and mini schema as set of sentence pairs, translated SQL queries will be mapped with other tables in schema, known as negative pairs. These three type of sentences will be combined into triple set of *anchor, positive, negative*. The goal is to minimizes the distance between translated SQL query and mini schema, while maximizes the distance between translated SQL query and the redundant tables in same schema. The loss function used here is *Triplet loss*.
- Phase 3 - Question to Mini-schema: The last phase would be to try to map the input to output, which is user question to mini schema. Similar to phase two, we created a triple set of *anchor, positive, negative*, where anchor is the user question. The *Triplet loss* is also used in this phase.

Once a well-fine-tuned base model was obtained on two versions of dataset, we further trained both versions using methods such as Few-shot, CoT, and Few-shot combined with CoT at both word and syllable levels to guide the model to learn inference behaviors and visualize the input and output of the task.

[2]https://huggingface.co/BAAI/bge-m3

[3]https://huggingface.co/spaces/mteb/leaderboard

6 VIETNAMESE EVALUATOR CONSTRUCTION

Due to previous research on SPIDER Yu et al. (2018), there are four levels of query complexity: EASY, MEDIUM, HARD, and EXTRA HARD, based on the provided definitions. Additionally, there are three metrics for evaluating model performance in the text-to-sql task:

- Exact Matching: checking whether the generated SQL query is the same as ground truth or not
- Component Matching: although not exactly the same because some components are permuted, the generated SQL queries are within the acceptable range. The metric verifies whether each component in the query matches the ground truth or not.
- Execute Matching: when checking the ground truth SQL statement against the predicted SQL statement, although they may differ, the execution results return the same outcome.

However, these metrics are only applicable to English schemas and SQL queries and cannot be directly applied to Vietnamese due to the presence of blank spaces in table or column names. For example, in Vietnamese SQL query `select ten, ten dem, ho from nhan vien`, the column name `ten dem` contains two words (`ten` and `dem`), similar to the table name `nhan vien`. Therefore, executing such queries may result in errors. To address this issue, our team attempted to modify the evaluator from SPIDER [4] by adding double quotes to each table or column name in SQL schemas and queries. With assistance from the `tokenize` function from the SPIDER GitHub repository, SQL queries were tokenized as shown in Figure 1.

```
select t1.id tài khoản , t2.tên tài khoản from giao dịch tài
 ↪  chính as t1 join tài khoản as t2 on t1.id tài khoản =
 ↪  t2.id tài khoản group by t1.id tài khoản having count ( *
 ↪  ) >= 4        customers_and_invoices
# tokens: ['select', 't1.id', 'tài', 'khoản', ',', 't2.tên',
 ↪  'tài', 'khoản', 'from', 'giao', 'dịch', 'tài', 'chính',
 ↪  'as', 't1', 'join', 'tài', 'khoản', 'as', 't2', 'on',
 ↪  't1.id', 'tài', 'khoản', '=', 't2.id', 'tài', 'khoản',
 ↪  'group', 'by', 't1.id', 'tài', 'khoản', 'having', 'count',
 ↪  '(', '*', ')', '>=', '4']
```

Figure 1: Result of the 'tokenize' function from the SPIDER evaluator.

```
select t1."id" "tài" "khoản" , t2."tên" "tài" "khoản" from
 ↪  "giao" "dịch" "tài" "chính" as t1 join "tài" "khoản" as t2
 ↪  on t1."id" "tài" "khoản" = t2."id" "tài" "khoản" group by
 ↪  t1."id" "tài" "khoản" having count ( * ) >= 4
```

Figure 2: Result after adding double quotes to target fields in names.

Figure 3: The result after each step of the process of adding double quotes.

Next, apart from sql keywords and sql symbols (such as $>=$, (, or $*$, or tokens identified as numeric values `isfloat==True`, these tokens are to be skipped accordingly. For tokens like 't1.id', the alias component (i.e., t1) must be identified and skipped. Note that in cases like 'nha truong.giao vien', the portion before the '.' is not always an alias. The remaining cases will have double quotes added at the beginning and end. The result is shown in Figure 2. Finally, postprocessing involves replacing (`" "`) with a single space ( ) and replacing double quotes (`""`) with a single one (`"`) for string data values.

[4] https://github.com/taoyds/spider/blob/master/evaluation_examples/README.md

Table 1: Benchmark on Exact Matching Accuracy.

	Model	Test
Vi-Word	**Ours_full-finetune**	0.5
	Ours_r4	9.8
	Ours_r64	12.6
	Ours_r128	**46.2**
Vi-Syllable	**Ours_full-finetune**	0.6
	Ours_r4	5.0
	Ours_r64	13.4
	Ours_r128	**47.0**

7 EXPERIMENTS

7.1 EXPERIMENTS SETUP

Every model trained to generate SQL queries directly from questions and schemas was trained using the pre-trained dataset for 1 epoch. The training was conducted without applying the LoRA technique, and with LoRA applied at ranks 4, 64, and 128; with a learning rate of 3.6e-5; max length of 1500; and batch size of 4 for both training and validation. The loss function used was Cross Entropy with causal masking.

After choosing effective LoRA rank, which is 128, the CodeLlama model was further trained with LoRA rank using the Chain-of-Thought method and also Few-Shot method on two version datasets: FullSchema and MiniSchema for 1 epoch; learning rate of 3.6e-5; max length of 5000; and batch size of 4 for both training and validation; the loss function was Cross Entropy with causal masking.

The CodeLlama model was also further trained with LoRA rank using Few-Shot Chain-of-Thought combination for 1 epoch; learning rate of 3.6e-5; max length of 6000; and batch size of 4 for both training and validation; the loss function was Cross Entropy with causal masking.

All experiments in this study were performed on an NVIDIA A100 80GB GPU.

7.2 **MAIN RESULTS**

From table 1, we see that full fine-tune and LoRA versions (r4, r64, r128) are compared based on Exact Matching accuracy for both tokenization methods:Vi-Word and Vi-Syllable. After conducting all training across, we have determined that using rank *128*. This decision is based on the Exact matching accuracy at both word and syllable levels (best result with accuracy of 47.0). Therefore, to optimize testing of future methods, we will proceed with training at LoRA rank *128* for subsequent steps.

In addition, it can be seen that the full-finetune method has a very low acurracy (0.5 for Vi-Word and 0.6 for Vi-Syllable). This is likely due to the fact that the data from the pre-trained set is not large enough to adjust all the parameters of the model, resulting in the model not being able to learn the necessary information effectively.

Based on the evaluation of query complexity levels (including Easy, Medium, Hard, and Extra Hard) according to previous works such as SPIDER Yu et al. (2018) and ViText2SQL Nguyen et al. (2020), as shown in Table 2, our team noticed that the Fewshot training method achieved the best exact-match accuracy across all four complexity levels, including both word and syllable levels. Specifically, the Easy and Extra Hard levels achieved the highest accuracy on word, reaching 93.2% and 62.5% respectively, while the Medium and Hard levels achieved the highest accuracy on syllable, reaching 81.9% and 74.4% respectively.

Based on the results in the table, we observed that although the Chain-of-Thought (CoT) training method did not perform well, achieving below 40% in exact match and lagging behind the current best method by about 13%, the Fewshot method achieved the best results on both word and syllable levels for Vietnamese, with 79.4% on the test set. Subsequently, we also experimented with combin-

Table 2: Benchmark on Exact Matching Accuracy.

	Model↓, Hardness→	Easy	Medium	Hard	Extra Hard
Vi-Word	**EditSQL_PhoBERT**	75.6	58.0	47.4	22.7
	IRNet_PhoBERT	76.8	57.5	47.2	24.8
	Ours_r128	76.7	58.0	47.4	22.7
	Ours_MiniSchema_r128	29.6	27.8	2.4	30.8
	Ours_Fewshot_r128	**93.2**	80.4	70.5	**62.5**
	Ours_MiniSchema_Fewshot_r128	45.9	31.2	46.8	27.5
	Ours_FewshotCoT_r128	84.2	63.3	50.6	33.2
Vi-Syllable	**EditSQL_PhoBERT**	75.1	56.2	45.3	22.4
	IRNet_PhoBERT	76.2	57.8	46.8	23.5
	Ours_r128	77.6	46.6	38.2	21.6
	Ours_MiniSchema_r128	29.6	27.8	2.4	30.8
	Ours_Fewshot_r128	92.5	**81.9**	**74.4**	61.1
	Ours_MiniSchema_Fewshot_r128	43.1	31.3	44.2	26.9
	Ours_FewshotCoT_r128	84.9	61.7	49.1	33.2

Table 3: Benchmark on Component Matching Accuracy.

	Component→ Model↓	SELECT	WHERE	ORDER BY	GROUP BY	KEY WORDS
Vi-Word	**EditSQL_DeP**	75.1	44.6	65.6	63.2	73.5
	IRNet_PhoBERT	83.3	61.8	72.5	67.9	80.6
	EditSQL_DeP	79.3	48.7	71.8	63.4	74.3
	IRNet_PhoBERT	84.5	59.3	76.6	68.2	80.3
	Ours_r128	74.6	59.4	76.1	68.0	77.1
	Ours_MiniSchema_r128	75.4	55.9	82.1	72.6	82.2
	Ours_Fewshot_r128	90.6	85.3	90.9	83.5	92.0
	Ours_MiniSchema_Fewshot_r128	78.2	60.4	82.6	72.8	82.4
	Ours_FewshotCoT_r128	80.8	71.2	83.9	73.8	84.9
Vi-Syllable	**EditSQL_XLM-R**	82.7	60.3	70.7	67.2	79.8
	IRNet_XLM-R	83.5	59.1	74.4	68.2	80.5
	Ours_r128	73.5	56.6	76.9	73.9	74.5
	Ours_MiniSchema_r128	76.7	57.6	83.7	71.7	83.0
	Ours_Fewshot_r128	**91.3**	**85.9**	**92.6**	**86.8**	**93.4**
	Ours_MiniSchema_Fewshot_r128	79.7	60.9	83.9	74.5	83.0
	Ours_FewshotCoT_r128	81.7	70.9	85.3	73.2	85.0

ing the two dataset version, FullSchema and MiniSchema; two training methods, Fewshot and CoT, and achieved better results than the current methods, but still lagged behind the Fewshot method by about 20%.

Furthermore, we can see that the MiniSchema method is giving quite low results when evaluating according to the Exact Matching Accuracy criterion. Specifically, with both tokenization methods (Vi-Word and Vi-Syllable), the MiniSchema method scores lower than other methods at all difficulty levels (Easy, Medium, Hard, Extra Hard), even when combined with Fewshot and FewshotCoT.

In addition to achieving good results on the exact matching metric, the model performs even better when considering the permutation of some components of the predicted query sentence and the target query sentence to have no effect on the correctness of the result. Therefore, from the Component Matching evaluation table 3. *Fewshot_r128* still achieves the highest accuracy, specifically at the syllable level, *Fewshot_r128* achieves higher accuracy than all other training methods with accuracy of: 91.3% for the SELECT component, 85.9% for the WHERE component, 92.6% for the ORDER BY component, 88.8% for the GROUP BY component and 93.4% for the KEYWORDS component.

Table 4: Benchmark on Execute Matching Accuracy.

	Model↓, Hardness→	Easy	Medium	Hard	Extra Hard	All
	Count→	425	779	403	301	1908
Vi-Word	**Ours_r128**	83.5	64.3	56.3	52.8	65.1
	Ours_MiniSchema_r128	81.6	73.9	61.7	65.4	71.7
	Ours_Fewshot_r128	**95.8**	86.0	83.4	**83.7**	87.3
	Ours_MiniSchema_Fewshot_r128	71.7	72.8	71.4	63.6	70.8
	Ours_FewshotCoT_r128	88.2	74.2	68.5	63.1	74.4
Vi-Syllable	**Ours_r128**	84.5	65.9	60.8	44.9	65.6
	Ours_MiniSchema_r128	81.2	73.8	62.0	64.8	71.5
	Ours_Fewshot_r128	95.1	**90.4**	**85.4**	76.4	**88.2**
	Ours_MiniSchema_Fewshot_r128	71.3	73.4	71.0	63.8	70.9
	Ours_FewshotCoT_r128	89.4	75.0	70.2	63.1	75.3

It is worth noting that although the MiniSchema method has a rather low exact matching score, it achieves a fairly high component matching score. Specifically, the *MiniSchema_r128* method achieves an average of 74.4% and *MiniSchema_Fewshot_r128* achieves 76.4%, higher than the *r128* average of 71.0%, only lower than *Fewshot_r128* with 90.0%.

Besides achieving good results on exact matching metrics 2 and component matching 3, for some queries that are not exactly the same and do not have the same components, if the query is executed, the results are as shown in the following table 4: The accuracy of Execute Matching is consistently higher than that of Exact Matching across all training methods. Specifically, the *FewShot_r128* method achieves the highest execution accuracy on both the word and syllable levels. In detail, the highest accuracies for Easy and Extra Hard queries at the word level are 95.8% and 83.7%, respectively; the highest accuracies for Medium and Hard queries at the syllable level are 90.4% and 85.4%, respectively. Furthermore, the *FewShot_r128* method achieves the highest average accuracy of 88.2% at the syllable level.

Based on our experiments, we have made the following key observations:

- **General training avoids overfitting**: Pre-training on a general text-to-sql dataset and then fine-tuning on specific case like the ViText2SQL dataset helps the model avoid overfitting. This is because the model only learns from the ViText2SQL dataset for a single epoch, compared to 10 epochs for the current state-of-the-art (SoTA) methods. Additionally, the model learns better because it has already seen a variety of query forms that can be generated through the general base.
- **Behavioral training should be based on the main task**: Training for Fewshot and CoT scenarios should be based on the dataset related to the main task. This is because tasks like Fewshot and CoT only change the way model receives and generates answer, but the main task that needs to be focused on is the prediction.
- **The Fewshot learning method is better than the Chain-of-Thought method in the *Code Domain***: This is an interesting observation from the experiments across all three metrics (exact matching, component matching, and execute matching) is that: $CoT < SQL - direct < FewShot + CoT < FewShot$ (A < B tells that method A is better than method B on the respective metric). This can be explained as follows: While it is intuitive to assume that interpreting a generated query would be highly effective, as this is a code generation model, interpretation can sometimes be inefficient, and the experiments have shown this to be the case. Additionally, the few-shot method alone outperforms all other methods.
- **The MiniSchema appoarch can be consider as an effective method for Text-to-SQL domain**: In table 4, although the MiniSchema method did not score high in exact matching, it gave quite impressive results in execute matching. Specifically, *MiniSchema_r128* scored an average of 71.5% and *MiniSchema_Fewshot_r129* scored an average of 70.9%, just behind *Fewshot_r128*.

- **Prediction shots must be highly relevant**: The effectiveness of the Fewshot learning method is largely due to the construction of the examples during training. Queries are trained with examples that have high semantic and query structure similarity, while ensuring fairness by preventing the test set from appearing in any training set and vice versa.
- **Consistency in training and inference shots**: We experimented with a progressive learning approach from easy to extra hard for Fewshot learning method. This approach involved randomly selecting a shot size from 1 to 3 and ordering the training samples by length from shortest to longest. While this method appeared promising, we observed during inference that the model sometimes generated additional virtual shots, up to 2 or 3. This behavior can be explained by the model's need to visualize the input and output, requiring a fixed shot size. Otherwise, during inference, the model would be uncertain about the number of shots required for sample generation. Consequently, we switched to training and inference with a fixed shot size of 3, achieving the best results on the ViText2SQL dataset.
- **Configuring max_length based on LoRA Rank**: The choice of max_length being 5000 instead of a larger number was motivated by several factors. Firstly, nearly 90% of the data falls within this range. Secondly, LoRA was used for training, which has a maximum rank of 128. Using a larger max_length would have made it impossible to recap enough information for learning, leading to performance degradation.

7.3 Limitations

Our proposed approach faces several challenges and limitations that warrant further investigation and improvement:

- **Effective MiniSchema approach**: We need to train a Retriever model that is highly accurate in identifying the real relationships between user queries and tables in the schema. Furthermore, choosing the appropriate threshold during evaluation also significantly affects the performance of a large language model.
- **Complex query handling**: The model's accuracy drops when dealing with complex queries involving multiple joins, aggregations, and grouping. This is attributed to the increased difficulty in capturing the intricate relationships and operations inherent in such queries.
- **Data validation**: The generated queries require additional validation to ensure that the specified values match the actual data stored in the database. This validation step is crucial to guarantee the integrity and correctness of the retrieved results.
- **Query optimization**: The generated queries may not be optimized for execution time and resource utilization. Developing heuristics to ensure query efficiency without compromising accuracy is essential for practical applications.

8 Conclusion

Contributions In this study, our team present a large language model (LLM)-based Text2SQL approach that not only outperforms the current state-of-the-art Vietnamese Text2SQL method by over 23% in terms of exact match (EM) but also supports query interpretation during inference. We also propose a potential approach of input schema filtering, which not only has a lot of potential for mining, but also helps reduce the number of input tokens, optimizing training and inference time. Additionally, we propose modifications to the Text2SQL evaluator to better assess the performance of Vietnamese queries.

Our LLM-based Text2SQL model is a multi-task model capable of generating text-to-SQL queries, explaining queries, summarizing tables, and interpreting query results. The model is trained on a newly constructed Vietnamese Text2SQL pre-trained dataset, comprising 240,000 samples. We employ various training strategies, including Fewshot, Chain-of-thought (CoT), Multi-CoT, Fewshot learning with CoT, and Schema filtering to train our LLM-based Text2SQL model.

In addition, our models are based on open-source models, so they have advantages in terms of safety and security, avoiding user information disclosure compared to using third-party APIs in real-world deployment.

Future works To further enhance the model's capability and address limitations, we propose several future research directions. Firstly, we intend to focus on improving the schema filtering method because we believe in its potential to optimize the training and inference space and improve the model performance. We plan to develop a more efficient training approach for the retriever model as well as improve the inference method.

Furthermore, we plan to extend our research to the field of conversational AI, leveraging graphs, SQL, and database data to enable the model to interact with users in natural language. This approach will provide a more intuitive and friendly way of interacting. To optimize the output query, we propose to develop a heuristic module, which uses various techniques to refine and improve the quality of the query, ensuring accuracy and efficiency. Finally, we will deploy this system into real-world applications, integrating with platforms such as SAP, IOC, and ERP, to demonstrate the effectiveness and applicability of the method in real-world scenarios.

ACKNOWLEDGMENTS

This research is funded by Viettel Business Solutions - Viettel Group.

REFERENCES

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report, 2023.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases, 2020.

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

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation, 2023.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder: may the source be with you!, 2023.

Xiping Liu and Zhao Tan. Divide and prompt: Chain of thought prompting for text-to-sql, 2023.

Anh Tuan Nguyen, Mai Hoang Dao, and Dat Quoc Nguyen. A pilot study of text-to-sql semantic parsing for vietnamese, 2020.

Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql, 2022.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers, 2021.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. Sparc: Cross-domain semantic parsing in context, 2019.

Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. Editing-based sql query generation for cross-domain context-dependent questions, 2019.

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

APPENDIX A: RETRIEVER MODEL

EVALUATION ON RETRIEVER MODEL

We evaluated the performance of the Retriever model in selecting the optimal MiniSchema with the following settings: selecting the top 3 tables in the Schema and applying different confidence thresholds: 60%, 65%, 70%, 75%, 80%, 85%, 90%.

By analyzing the performance of the Retriever model [4], we found that setting the confidence threshold too low (below 65%) will result in the model fetching too many unnecessary data tables, reducing the processing efficiency. On the other hand, if the threshold is set too high (above 75%), the model will become too demanding and cannot find any suitable tables. To solve this problem, we will experiment with different confidence thresholds (65%, 70%, and 75%) combined with selecting the top 3 tables with the highest scores. Then, we will use the LLM model to evaluate the quality of the results returned from each threshold and from there make a final decision on the optimal threshold.

PERFORMANCE OF LARGE LANGUAGE MODEL WITH RETRIEVER

Table 5: Benchmark on Execute Matching Accuracy on MiniSchema_r128.

	Easy	**Medium**	**Hard**	**Extra Hard**	**All**
bge_top3	81.2	73.8	62.0	64.8	**71.5**
bge_p65	73.6	73.4	65.5	64.5	70.4
bge_p70	77.6	72.9	60.5	59.8	69.3
bge_p75	75.3	58.9	51.1	41.9	58.2

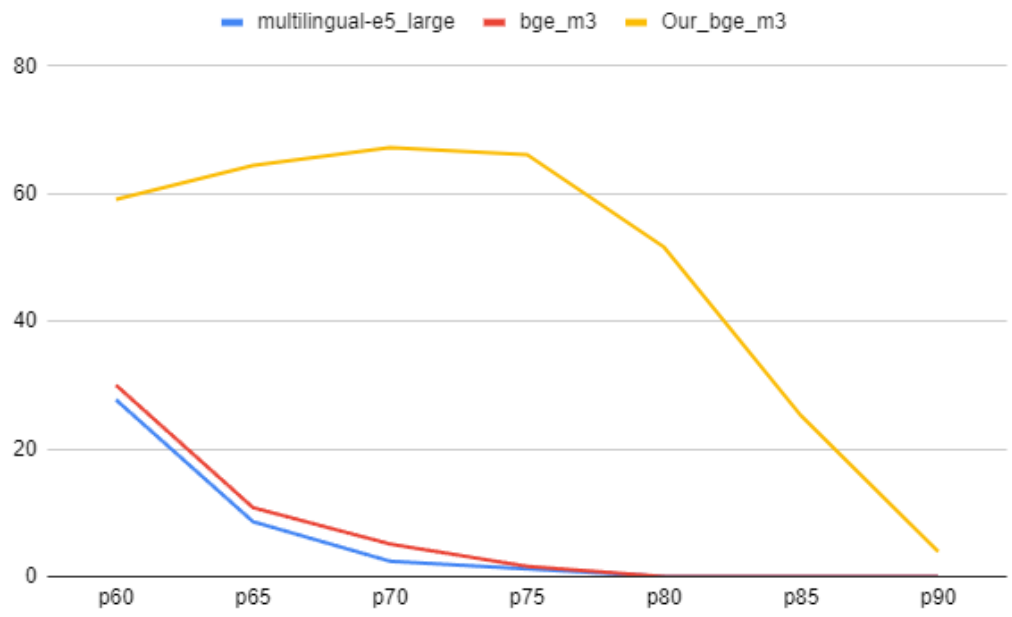

Figure 4: Recall@1 of Multilingual-e5 and BGE Models Across Threshold

In table 5, we can see that setting threshold of *p65* gives the best performance in execute matching, with an average score of 70.4%. This result is nearly equivalent to the selecting top 3 most similar tables method, which achieved 71.5% on an average. Through these findings, we decided to adopt the *top 3 selection* strategy as the output standard for our retriever method.