Will the real SQL translator, please stand up?

Tyler Cranmer, Jayant Duneja

University of Colorado Boulder

Abstract. The transformation of natural language queries into SQL commands is crucial for leveraging Electronic Health Records (EHRs), making healthcare data more accessible and actionable. This paper presents a novel approach that fine-tunes Small Language Models (SLMs) specifically for the task of text-to-SQL conversion within the healthcare domain. Unlike traditional methods that utilize large language models (LLMs) with in-context learning without altering the models' intrinsic weights, our approach modifies a subset of the SLM's parameters to tailor it to specific EHR text-to-SQL retrieval tasks. By incorporating fine-tuning and in-context learning techniques, we enhance the model's accuracy and efficiency in generating SQL queries directly from medical professionals' natural language questions. We demonstrate through experiments that our fine-tuned models not only outperform the baseline models in accuracy but also maintain high efficiency, making them viable for real-world applications where computational resources are limited. This research contributes to the field by providing a scalable, cost-effective solution for data querying in EHR systems, potentially transforming how healthcare professionals interact with and utilize patient data.

1 Introduction

The motivation behind our work is rooted in the transformative potential of EHRs, which serve as the digital analogs to traditional patient paper charts. EHRs offer real-time, patient-centered records that can securely disseminate information to authorized users without delay. Despite their value, the ability to explore beyond pre-set queries requires a specific set of technological skills, such as proficiency in SQL. To address this, we propose the development of an Artificial Intelligence (AI) interface designed to simplify the process of accessing EHR data. This interface, which would utilize a precise text-to-SQL model, aims to convert natural language questions into SQL queries efficiently. Such a tool would significantly benefit healthcare professionals and researchers by making complex data queries more accessible and intuitive. An additional consideration driving our initiative is the cost-effectiveness comparison between employing few-shot prompting with models like GPT-4 and fine-tuning smaller models, with the objective to determine the most economically viable option both presently and looking ahead.

The recent influx of large language models equipped with text-to-SQL capabilities, including every model released since Chat-GPT or GPT 3.5, has been

notable. Currently, GPT-4 outperforms other decoder generative models like GPT 3.5 or T5 in terms of efficiency. However, a significant limitation arises due to their training on generic datasets, resulting in their generated SQL query responses being contingent on an assumed database schema. To navigate this challenge, one strategy involves employing few-shot prompting, wherein the model is fed example schemas and queries to better tailor its generated queries to specific natural language questions. We intend to thoroughly evaluate the efficacy of this approach across various models, leveraging benchmarks for GPT 3.5 and T5 provided by the shared task, to pinpoint and address the shortcomings of current solutions.

Our proposal introduces a novel approach by leveraging SLMs, which have recently gained popularity due to their finetuning flexibility and significantly fewer parameters compared to their larger counterparts. For instance, whereas Chat-GPT boasts roughly 175 billion parameters, the newly released Llama-2 open-source models have a markedly lower count of about 7 billion parameters, illustrating a substantial reduction in scale. The advent of Parameter-Efficient Fine-Tuning (PEFT) techniques such as Lora and Qlora has made it feasible to train these SLMs within the existing architectural framework. Our aim is to deepen our understanding of these methods, which we believe will equip us with the knowledge to more effectively manipulate these models. By finetuning SLMs and evaluating their performance against the few-shot prompting method, we anticipate that our approach will not only present a new avenue for solving this research problem but also enhance our grasp of existing methods and tools, ultimately offering a superior solution.

2 Related Work

2.1 RAG-Based Approaches:

Recent advancements in Text-to-SQL technologies highlight various approaches to enhancing the performance of LLMs in healthcare and epidemiological data applications. The PET-SQL framework introduces a two-stage process starting with a novel prompt representation that includes schema and randomly sampled cell values, followed by schema linking and cross-consistency checks across different LLMs, achieving state-of-the-art results [1]. Similarly, the integration of text-to-SQL generation with retrieval augmented generation (RAG) addresses the intricacies of medical terminology and complex SQL query structures by incorporating medical coding, thus improving the contextual awareness of model outputs [2]. Another approach systematically evaluates prompt engineering techniques and introduces DAIL-SQL, combining the strengths of various methods for enhanced accuracy, especially emphasizing token efficiency in prompt engineering and supervised fine-tuning of open-source LLMs [3].

In contrast, our research focuses on fine-tuning SLMs specifically for text-to-SQL tasks within electronic health record systems. Unlike the broader approaches that primarily modify input prompts or integrate complex retrieval mechanisms, our method involves directly altering the model's weights through

fine-tuning. This allows for a tailored adaptation to the nuances of EHR-based text-to-SQL retrieval tasks, presenting a resource-efficient solution that leverages the intrinsic flexibility of SLMs. Our fine-tuning approach addresses specific medical querying requirements, offering a distinctive and adaptable alternative to the existing methodologies that depend heavily on large pre-trained models and elaborate external adjustments.

2.2 In-Context Learning Based Approaches

The fundamental distinction between in-context learning and our proposal for fine-tuning SLMs hinges on the strategies for augmenting the performance of these models. In-context learning consists of prompt engineering, a method whereby targeted examples or directives are presented to LLMs at inference time without modifying the models' intrinsic weights. This technique is utilized in applications that involve Few-Shot Text-to-SQL tasks [4]. The aim is to enhance the model's capability to translate natural language instructions into SQL queries by supplying a limited number of example outputs within input prompts. This approach does not alter the pretrained model's weights, thus the LLMs' architecture remains intact [5].

We will be fine-tuning a subset of the SLM's parameters by continuing the training phase on a curated dataset that is more narrowly focused on our specific EHR text-to-SQL retrieval task. This process modifies the model's weights, thereby tailoring it more closely to our specific text-to-SQL retrieval task. With that said, this process requires careful handling to prevent overfitting.

2.3 Seq2Seq Based Approaches

These approaches utilized Seq2Seq architectures, which feature an encoder that transforms variable-length sequences into a fixed-length encoding and a decoder that reconverts this encoding into another variable-length sequence. Employed in tasks involving complex sequential data generation, these models enabled end-to-end learning before the advent of decoder-based large language models. They are notably referenced in studies [6] and [7], demonstrating their utility and effectiveness.

3 Methods

3.1 Dataset

Original Data: The dataset we will be using comes from the NAACL - Clinical NLP 2024 Shared Task, which comprises 7,454 natural language instruction queries and their expected SQL queries. The data is divided into a 70% training, 15% validation, and 15% test set. Each training example is a JSON object that includes a sample identifier (ID) and a natural language question, which can be either answerable or unanswerable given the MIMIC-IV schema. Each label will

have the corresponding identifier and the SQL query, or 'null' if the query is invalid. The database table schema to which the output SQL queries will refer follows the same style as the Spider Dataset [8].

Augmented Data: To increase the dataset size, we used GPT-4 to rewrite the natural language questions provided by the shared task. The original dataset was expanded by four times, adding a total of 20,539 natural language queries and their respective SQL query pairs. To ensure the preservation of important data from the original query, we prompted GPT-4 with the following prompt:

```
For each of the following medical queries, please generate
four alternative phrasings. The alternative queries should ask
the same question in slightly different ways, focusing on
variations in wording, structure, and formality, while keeping
the essential medical and contextual information intact.
```

This prompted resulted in 4 new but similar queries that were then mapped to the original sql label.

Original natural language instruction:

- Tell me the minimum respiratory rate in patient 10021118 during the first ICU visit.

Four new augmented examples:

- Provide the minimum respiratory rate recorded for patient 10021118 on their first ICU admission.
- Can you report the lowest respiratory rate observed for patient 10021118 during their first visit to the ICU?
- Please inform me of the minimum respiratory rate recorded for patient 10021118 during their first stay in the ICU.
- What was the lowest respiratory rate for patient 10021118 during their initial ICU admission?

3.2 Instruction Fine-Tuning

Fine-tuning these newer large language models is different from fine-tuning older models like Bert. For these older models, we would mostly freeze the Bert parameters, add layers on top of the existing model and then train those additional layers. For these decoder style models however, we need to provide the training data in an instruction based format.

System Prompt: For providing the system prompt to the model, we tried a few different methods. The first method was one which was provided to us by the shared task organizers, which first gave the model an instruction to generate

a SQL query for a given natural language question, and then gave the details for all the tables in the database. For giving the details, the shared task organizers just provided a string of the table name, and then a python list of all the columns that are present in the table.
The second variation of the prompt formatting we tried had 2 changes. First, we added the keyword

```
#diction: sql
```

which we found out that it might be a specific keyword which helps improving the performance of llama. The second change we made instead of providing all the tables that are present in the database, we only provided the specific tables that were mentioned in the query. We were able to find only the specific tables mentioned in the query using a simple string regex. The final variation we used which provided us with the best results was using the SQL CREATE Table statements for the different tables that were mentioned. Hence, instead of providing the details as a python list, we provided them in SQL syntax.

Prompt Formatting: Initially, we decide to use the prompt format that was used by the Alpaca Paper to finetune the model [9] . But while using this method, the model was not able to learn the end of sequence token efficiently which led to results where the model was not able to learn how to stop generating a sequence.
Consequently, we reverted to using the exact prompt employed in the llama paper for fine-tuning. The rationale behind this decision is to ensure consistency between the prompt used during training and fine-tuning, optimizing the model's learning process.
The exact format that we used for training and inference can be found in figure 1.

3.3 Hardware Requirements

We required NVIDIA A100 GPUs for training due to their substantial system memory, as older GPUs were inadequate. Acquiring these GPUs was challenging and costly at 50$ out of pocket. Consistently securing an A100 proved difficult, sometimes leading to long wait times before training. While training without these GPUs using Peft and Lora techniques is possible, combining the original model with adapters for inference necessitates the highest-grade GPUs due to loading constraints. The Llama2 model with 4-bit quantized parameters and a batch size of 8 utilized 28GB of GPU memory, while Llama3 used 37GB under the same settings.

3.4 Parameter Efficient Fine-Tuning

Certain estimates have mentioned that ChatGPT has close to 175 billion parameters. We can only imagine how many parameters GPT-4 would have but

Fig. 1. Prompt Formatting for LLama-2 and Llama-3

we can see that fine tuning models of such large sizes requires a large amount of enterprise hardware. Parameter efficient fine-tuning aims to train a much smaller number of extra model parameters while freezing most of the parameters of pre-trained LLMs. One such PEFT method which we plan to use is known as LoRA(Low-Rank Adaptation).

Lora aims to represent the weight updates for any layer of the original model with update matrices with the help of low-rank adaptation. The original weights of the model are kept frozen and with the help of this, the number of parameters we train is significantly lower than the original number of parameters. To get the final results, the original weights and the update weights are combined.

Quantization: To reduce the memory footprint during model training, we implemented quantization of the SLMs matrix weights. This technique, as described in the QLoRA paper, involves reducing the precision of the weight parameters in the pre-trained LLM from the typical 32-bit format to 4-bit precision. Quantization helps to optimize memory usage without significantly compromising performance. Specifically, quantization works by mapping the continuous range of weight values to a smaller set of discrete values, effectively compressing the model's data size. [10]

By using the QLoRA method, we were able to fine-tune approximately 0.59% of Llama3 and 0.95% of Llama2 parameters, which further demonstrates the efficiency of our approach. This fine-tuning process allowed us to maintain high model accuracy despite the reduction in parameter precision.

3.5 Small Language Models

SLMs are smaller versions of their LLM counterparts. They have significantly fewer parameters, ranging from a few million to a few billion, in contrast to LLMs, which can have parameters ranging from hundreds of billions to trillions. The difference in size between the two models offers several advantages. SLMs require less computational power and memory, making them more efficient and suitable for running inference on smaller devices and in edge computing environments. With the reduced need for computation, there is higher accessibility for a broader range of developers and organizations. Due to the lower resource requirements of SLMs, developers can more easily fine-tune their models for specific domains or tasks.

4 Expirements

4.1 Main Purpose

We plan to finetune two open source large language models : llama-7b released by Meta which contains 7 billion parameters and llama-3-8b, which is an open source model released by Meta which contains 8 billion parameters.

We will compare our fine tuned models against the base models with the help of the evaluation metrics used for this shared task. We also plan to compare the performance of our fine tuned models against existing Large Language Models like GPT-3.5 and GPT-4, using in context learning approaches already mentioned in the shared task.

4.2 Training

For training the model, we used the Pytorch framework and other libraries like the peft library which helped us define the LoraConfig which we used,loading the quantized model and merging the adapters back with the original model. We used the bitsandbytes library to define the quantization configuation that we needed. Apart from that, we used the SFTT Trainer(Supervised Fine-Tuning Trainer) for fine-tuning the model. For us, training took 2 - 2.5 hours per epoch. Due to this, we were only able to train different variations of the model for only a single epoch. The loss curves for LLama-2 and LLama-3 training can be seen in figure 2.

4.3 Editing the Attention/Loss Mask:

We made significant edits to the model's loss and attention masks to enhance fine-tuning performance. One key change was introducing a new pad token, discontinuing the use of the end-of-sequence (eos) token as the pad token, and assigning a specific index to the pad token. The attention mask now assigns a value of 1 to actual tokens and 0 to padding tokens, directing the model's

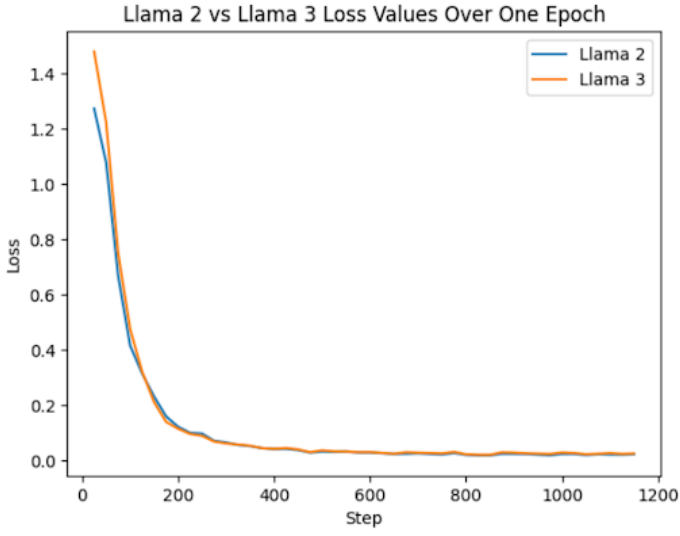

Fig. 2. Training Curves for LLama-2 and LLama-3

attention effectively during processing. Conversely, the loss mask specifically focuses on "Generated SQL Query" tokens, assigning a value of 1 to prioritize learning from them while disregarding other tokens and padding them with 0 during training. An illustration of the edits we made can be found in the figure 3.

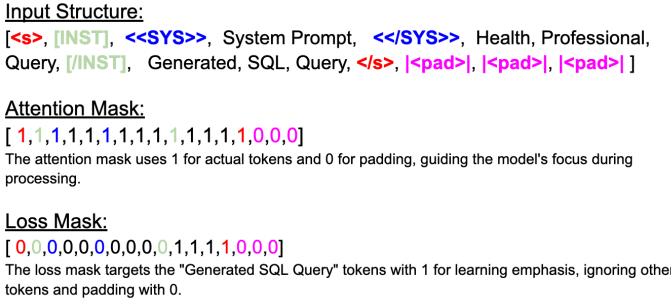

Fig. 3. Edits to the Loss and Attention Masks Made

4.4 Hyper-Parameters

LoRA: In our LoRA model configuration, we set the rank to 64m which determines the complexity of the adaptation and directly influences how much the

pre-trained weights are modified. The Alpha parameter was set to 16, controlling the learning rate adaptation. Additionally, we implemented a dropout rate of 0.1 to help prevent overfitting by randomly omitting units during the training process. Which helped create a more generalized model.

Quantization: For our bits and bytes quantization settings we enabled 4-bit precision, which significantly reduced the model's memory requirements for training. The quantization type was set to 'nf4'. The computation during quantized operations were carried out using 'float16'. Finally we decided to opt out of the double quantization hyper parameter to ensure that the model still had some precision to its quantize values.

Supervised Fine-tune Trainer: To train the model, we configured the Supervised Fine-Tune Trainer (SFFT) with a range of hyperparameters to ensure effective learning while optimizing computational efficiency. The model was fine-tuned for 1 epoch, which constitutes a single pass through the dataset. Both the training and evaluation phases utilize a batch size of 16. Gradient accumulation steps were set to 1, allowing for straightforward updating of model weights. We constrained the maximum gradient norm to 0.3 to prevent exploding gradients. Cross-entropy loss was used as the loss function to effectively measure the model's performance against the actual outcomes.

The learning rate was set at 0.00002, and we used a cosine learning rate schedule for gradual adjustments throughout the training process. The optimizer was set to adam 32bit, including a weight decay of 0.001 to avoid over fitting. A warm-up ratio of 0.03 was used to increase the learning rate from zero to the initially set value, which helped achieve more stable convergence. The data was grouped by length to optimize padding and processing time. Finally, the maximum sequence length was set to 1024.

5 Results

5.1 Evaluation Metrics

The Shared Task does not use any traditional NLP metrics like the Rouge scores or the Bleu scores for comparing the SQL queries against one another. The Shared task actually runs the generated SQL queries on a mimic database and then compares the results against each other. Since the shared task has ended officially, we plan to run the scorer on our own to evaluate the performance of our model.

The Shared Task uses a reliability score metric for ranking the results. The `reliability_score` function evaluates the accuracy of predictions by comparing predicted results against real results, assigning scores based on correctness, absence of prediction, or incorrectness. The penalize function then modifies these scores, specifically reducing the value of incorrect predictions (-1 scores) by a

penalty factor, and calculates the average of these adjusted scores. This process quantifies the reliability of predictions and applies penalties to inaccuracies, offering a numerical measure of prediction performance.

5.2 Post-Processing Results

For our dataset, there were certain natural language questions for which the corresponding SQL query is "null". This is due to the fact that those queries are not related to electronic health records and that is the reason why they have been replaced by "null" in the training set. For the models we trained, in some cases, the model still generated queries for these types of questions which then led to errors once we ran the queries against the mimic database. Due to this, the file that had the results had a lot of values which were equal to "error_pred".

Thus, in the tables shown below, the post-processed results shown are the results we got once we replaced these "error_pred" values with the value "null", since the model should not have generated a query for those questions in any case.

5.3 Results

We noticed a few interesting things in the results that were generated.

First of all, we noticed with the prompt that was used by the organizers of the shared task for a particular baseline, GPT 3.5 and GPT-4 both performed somewhat similar, even after the post processing that we attempted. This shows that the prompt used plays a large part in the performance of the model.

The second interesting trend noticed is that for LLama-2, the performance before the post-processing is poor and is similar to the GPT models. However, once I applied the post-processing, the accuracy shoots up to about 50%. This shows that LLama-2 was not able to learn when it should return a null query effectively.

The final observation for these models is that for LLama-3, it was naturally able to learn the situations in which it was supposed to generate null queries. That is why even we completed the post-processing, the performance did not improve greatly. This shows that Llama-3 is much more flexible than its predecessor and is able to understand different instructions.

The results generated can be found in the table below:

6 Conclusion

Our research demonstrated the effectiveness of fine-tuning small language models for the specific task of converting natural language queries into SQL commands

Model	Accuracy %	Accuracy % after Post-Processing
GPT-3.5*	27.0	28.3
GPT-4*	30.7	31.6
Llama 2 7B (Fine Tuned)	31.8	51.5
Llama 3 8B (Fine Tuned)	59.6	60.6

Table 1. Model Accuracy Before and After Post-Processing

within the context of electronic health records. By adopting our approach to fine-tuning small language models, healthcare companies that need to run local AIs due to privacy concerns can do so with higher accuracy and efficiency compared to traditional large language models that solely rely on in-context learning. This research brings AI applications in healthcare closer to practical implementation by making data queries more accessible to medical professionals without SQL expertise.

While this technology stands to significantly enhance efficiency and accessibility in healthcare data management, it also presents potential challenges, such as the displacement of jobs. Data analyst who traditionally handle SQL queries and data interpretation, may find thier roles evolving or diminishing as AI tools become more capable of performing complex data queries autonomously. It's important for organizations or companies to proactively manage this transition, offering retraining and reskilling opportunities to affected employees. By doing this, companies can mitigate the impact of job displacement and also leverage the existing expertise of analysts to over see and improve AI operations

References

1. Li, Z., Wang, X., Zhao, J., Yang, S., Du, G., Hu, X., Zhang, B., Ye, Y., Li, Z., Zhao, R., et al.: Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. arXiv preprint arXiv:2403.09732 (2024)
2. Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J.: Text-to-sql empowered by large language models: A benchmark evaluation. arXiv preprint arXiv:2308.15363 (2023)
3. Ziletti, A., D'Ambrosi, L.: Retrieval augmented text-to-sql generation for epidemiological question answering using electronic health records. arXiv preprint arXiv:2403.09226 (2024)
4. Nan, L., Zhao, Y., Zou, W., Ri, N., Tae, J., Zhang, E., Cohan, A., Radev, D.: Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies. arXiv preprint arXiv:2305.12586 (2023)
5. Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Lin, J., Lou, D., et al.: C3: Zero-shot text-to-sql with chatgpt. arXiv preprint arXiv:2307.07306 (2023)
6. Zhong, V., Xiong, C., Socher, R.: Seq2sql: Generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103 (2017)
7. Wang, P., Shi, T., Reddy, C.K.: Text-to-sql generation for question answering on electronic medical records. In: Proceedings of The Web Conference 2020. (2020) 350–361

8. Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al.: Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. arXiv preprint arXiv:1809.08887 (2018)
9. Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., Hashimoto, T.B.: Alpaca: Harnessing the power of simplification for model efficiency. (2023)
10. Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: Qlora: Efficient finetuning of quantized llms (2023)