Leveraging Open-Source Language Models for Privacy-Preserving Text-to-SQL Generation in Electronic Health Records

Grace Chan
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
gracechan668@gmail.com

Po-Chih Kuo
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
kuopc@cs.nthu.edu.tw

Abstract

Electronic Health Records (EHRs) are valuable sources of clinical information, yet querying them often requires specialized knowledge of structured query languages. Large Language Models (LLMs) enable more intuitive access through natural language queries; however, achieving reliable performance often requires domain-specific fine-tuning, and proprietary LLMs may raise privacy concerns because sensitive data must be transmitted to external servers. To address these challenges, we propose a fully open-source, locally deployable framework that integrates two complementary models, Llama-Coder-13B and DeepSeek-Coder-6.7B, fine-tuned using schema-aware prompts. Their outputs undergo confidence-based filtering and rule-based aggregation, leveraging the strengths of both models to improve accuracy and reliability. Evaluated on the EHRSQL 2024 shared task using the Reliability Score (RS), our approach achieves an RS(0) of 82.69 and an RS(10) of 44.99, outperforming other open-source baselines. Although a performance gap remains compared to proprietary LLMs, our system offers a practical privacy-preserving alternative while maintaining competitive performance among open-source approaches.

Keywords: Electronic Health Record, text-to-SQL, Large Language Models

1 Introduction

Electronic Health Records (EHRs) serve as databases that document patients' medical histories within healthcare institutions. EHRs are usually stored as structured data, where they adhere to a predefined format and can be readily stored, accessed, and analyzed using traditional database systems. Storing in such a way allows the EHR to be standardized and consistent, making it ideal for quantitative analysis and population health studies. A previous study (Vuokko et al., 2017) highlights the importance of completeness and interoperability in structured EHR data, which are essential for healthcare organizations to deliver high-quality, efficient, and integrated care.

Although these datasets provide valuable clinical information, accessing and analyzing their content typically requires proficiency in query languages such as SQL, as researchers must define and extract relevant patient cohorts. However, due to the complex and often heterogeneous structure of medical databases, crafting SQL queries can be tedious and error-prone, often requiring assistance from data specialists. Following the rise of Large Language Models (LLMs), this process may become more accessible, as LLMs have the potential to translate natural language queries into structured database commands. However, even though LLMs demonstrate strong performance in general coding tasks, their ability to generate accurate SQL queries without fine-tuning remains limited in the medical domain, where schema complexity and domain-specific knowledge pose significant challenges (Chadha et al., 2025).

To improve LLM performance on this task, fine-tuning on medical data is often necessary. However, using proprietary LLMs raises privacy concerns, as uploading sensitive data to external servers may compromise patient confidentiality. Moreover, hospitals typically operate on closed networks without internet access, making it infeasible to rely on cloud-based proprietary models. These constraints underscore the importance of developing effective and privacy-preserving SQL generation systems using open-source LLMs that can be deployed locally within healthcare environments.

In this study, we aim to develop a reliable SQL generation system using only open-source LLMs. Our approach combines two fine-tuned LLMs with contrasting generation behaviors to balance preci-

Proceedings of the 38th Conference on Computational Linguistics and Speech Processing (ROCLING 2026), pages 1–11
November 6-7, 2026 ©2026 Association for Computational Linguistics

sion and exploration in text-to-SQL:

1. **Model A:** Conservative — cautious, precise, and avoids risky outputs.

2. **Model B:** Daring — exploratory, ambitious, with a higher risk-reward trade-off.

Both models are fine-tuned and prompted using the same strategy, which provides schema information to improve their understanding of the database structure. After generating their predictions, a confidence filter is applied to discard outputs with low total log probability, ensuring that only high-confidence results are retained. A rule-based mechanism then merges the results, leveraging the strengths of each model while allowing them to cross-validate one another.

We validate our system in the EHRSQL 2024 shared task (Lee et al., 2024c), where it outperforms other open-source baselines in terms of correctness, while also demonstrating improved reliability.

In summary, our study contributes a method for enhancing the reliability of text-to-SQL systems using only open-source models, enabling hospitals to fine-tune and deploy these systems locally, and thereby providing easier EHR access to medical professionals.

The main contributions of our research are:

1. We develop an open-source LLM-based SQL generation framework for EHRs that achieves competitive performance compared with existing open-source approaches.

2. We propose a dual-model framework that combines two LLMs with complementary generation styles to effectively balance precision and diversity in text-to-SQL generation.

2 Related Works

A common strategy for building a reliable SQL generator is the pipeline approach. Instead of addressing the complex task of SQL generation in a single step, this method decomposes the process into multiple stages, with each stage handling a simpler and more focused subtask. For example, (Lee et al., 2024a) divides the process into three phases: schema linking, query generation, and result selection. This staged approach allows the language model to concentrate on one task at a time, thereby improving performance and reducing overall complexity. In more challenging scenarios, such as when the system must determine not only how to generate accurate SQL but also whether the user's question is answerable, Previous works (Somov et al., 2024; Thomas et al., 2024) design their systems to first assess answerability, followed by the generation stage, and finally a SQL inspection phase. However, increasing the number of stages can lead to error propagation, where mistakes in earlier stages negatively affect subsequent ones (Thomas et al., 2024).

A previous study (Gundabathula and Kolar, 2024) adopts an ensemble approach that combines multiple LLMs to enhance system reliability by reducing errors. Their experiment demonstrates that this method is effective when using proprietary LLMs, where a combination of models maintains more consistent reliability compared to a single GPT-4 model. However, its effectiveness with open-source models remains an open question and requires further investigation.

To enhance the capabilities of LLMs, some studies have focused on autonomous data generation. A previous study (Jo et al., 2024) demonstrated that introducing new training data during the self-training process can improve LLM performance on challenging cases, thereby increasing reliability in text-to-SQL tasks. Even though data augmentation can enhance the performance of a model, it requires careful analysis of the model's strengths and weaknesses and may necessitate additional effort to ensure the quality and relevance of the generated data.

3 Shared Task and Dataset

The dataset we will be using for our research is the EHRSQL 2024 dataset (Lee et al., 2024c). It is a recent benchmark designed to support the generation of reliable SQL queries in response to natural language questions posed on the MIMIC-IV DEMO database. By involving healthcare professionals in the question design process, the dataset captures a diverse range of queries that reflect real-world clinical information needs. A key feature of this shared task is the inclusion of unanswerable questions, which requires systems not only to generate accurate SQL but also to identify when a question cannot be answered, thereby avoiding the production of misleading or incorrect outputs.

The shared task of this dataset requires the system to not only generate accurate SQL queries in

cell1	Total Samples	% Unanswerable
Train	5124	8.78%
Valid	1163	19.95%
Test	1167	19.97%

Table 1: EHRSQL-2024 Dataset Statistics.

response to user requests but also to identify unanswerable questions and appropriately abstain from answering. Table 1 shows the number of answerable and unanswerable questions in the EHRSQL 2024 dataset.

4 Methodology

We fine-tuned two open-source models for this task using all the training data provided in the EHRSQL 2024 dataset. A set of filtering mechanisms is then applied to remove answers with low confidence and invalid SQLs before entering the ensemble stage. The flowchart of the whole system is shown in Figure 1.

4.1 Finetuning LLMs

For this study, we fine-tuned two different open-source LLMs: DeepSeek-Coder-6.7B (Guo et al., 2024) and LLaMA-Coder-13B (Touvron et al., 2023). Both models were fine-tuned and prompted in the same manner, using the prompt shown below. Inspired by (Jo et al., 2024), the prompt was designed to include information about the database schema, enabling the LLMs to better grasp the structure and contents of the database. Further details about the finetuning parameters will be mentioned in sec: experiment.

You are given SQL table schema and the question. Generate the SQL query for the following question. Note that you should generate 'null' if the question cannot be converted to SQL query using the given information.
SCHEMA
[patients]
row_id (int)
subject_id (int)
gender (varchar(5))
. . . . . .

Foreign key relationship
admissions.subject_id = patients.subject_id
diagnoses_icd.hadm_id = admissions.hadm_id
. . . . . .

##Question: "Throughout this year, what are the top five most common drugs prescribed during the same hospital encounter to female patients aged 50s after being diagnosed with epilepsy, unspecified, not intractable, without status epilepticus?"

##SQL:

4.2 Confidence Filtering

When testing our LLMs on the validation data, as shown in Appendix B, we notice that LLaMA-coder-13B tends to be more conservative, preferring to abstain if it is unsure about the question. On the other hand, Deepseek-coder-6.7B tends to be more daring in SQL generations, as it is more willing to return a SQL than to abstain from answering. Even though it helps in generating more correct results for the user, it also causes Deepseek-coder-6.7B to have a higher error rate as it rarely abstains from its answer when facing uncertain questions.

Since token-level log probability serves as an effective indicator of the model's confidence in its predictions, we follow (Kim et al., 2024) and (Thomas et al., 2024) by adding confidence filtering to reduce the error rate for Deepseek-Coder-6.7B. Removing the SQL if the total log probability is lower than the threshold, making sure all low-confidence SQLs are not returned to the user. We use the validation data to identify an optimal threshold that reduces the error rate of Deepseek-Coder-6.7B while still preserving its exploratory capabilities. Further details about the process will be discussed later on.

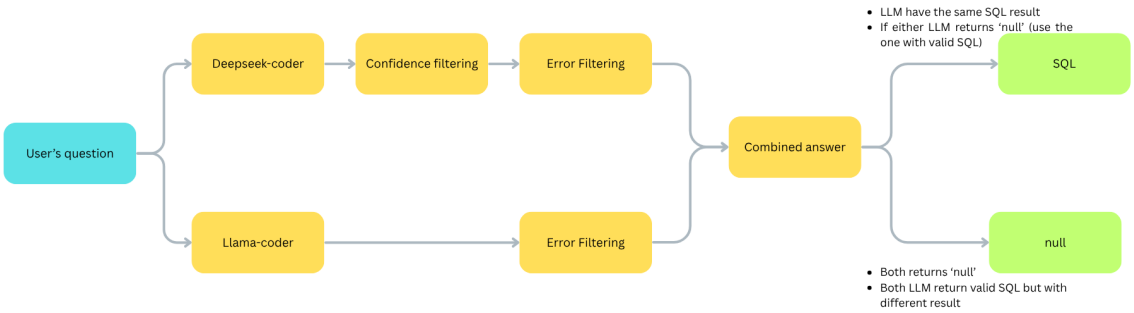

Figure 1: Overview of the proposed framework. User questions are processed by two fine-tuned open-source LLMs, Llama-Coder-13B and DeepSeek-Coder-6.7B. Confidence filtering and execution-based validation are applied before a rule-based ensemble module combines the outputs to generate the final SQL query or abstain with null.

4.3 Execution-Based Filtering

After the confidence filtering, we then execute the result obtained by both LLMs. When evaluating our system on the testing data, if the SQL cannot be executed on the MIMIC-IV-DEMO database or returns zero results, it will be excluded, and 'null' will be returned, following the dataset's format.

However, when the system is used in an application, these cases will not return 'null' but instead provide the reason why no results are given.

4.4 Ensembling LLMs

In order to preserve the strength of both models, while also allowing them to cross-examine each other's answers, we implement the combination rules as below, combining both models' answers and returning the final result:

1. **Both SQL yield the same results:** Returns the results
2. **One of the models returns 'null' while the other is a valid SQL:** Return the result of the valid SQL.
3. **Both SQL are valid but yield different result:** Returns 'null'
4. **Both model returns 'null':** Returns 'null'

5 Experiments

5.1 Finetuning Parameters

We use LoRA to finetune both Deepseek-coder-6.7B and LLaMA-coder-13B. Deepseek-coder-6.7B was finetuned for 5 epochs using $r = 16$ and a learning rate of $3 * 10^{-4}$ on NVIDIA RTX 3090. On the other hand, LLaMA-coder-13B was finetuned using AMD Instinct MI300X for 8 epochs with $r = 8$ and learning rate of $3 * 10^{-4}$.

5.2 Confidence Filtering Threshold

We evaluate our system using the validation set to determine the optimal threshold for confidence filtering in LLaMA-Coder-13B. Specifically, we compute the RS scores across various thresholds and discard those configurations where RS(0) falls below 50. The model achieves its best RS(0) and RS(10) scores without the confidence filtering, with RS(0) = 57.52 and RS(10) = 21.41. Although the best RS(N) is observed at threshold = -21.2, with RS(N) = -3849.87, we decided not to apply confidence filtering to LLaMA-Coder-13B, as our primary evaluation metric is RS(10). Further details about the result of each threshold will be shown in Figure 3, Figure 4 and Figure 5 in Appendix B.

A similar evaluation was conducted for Deepseek-Coder-6.7B, with thresholds where RS(0) falls below 70 being removed. A higher threshold was used for Deepseek-Coder-6.7B to preserve the model's exploratory capabilities, aiming to complement the more conservative behavior of LLaMA-Coder-13B. Among the remaining candidates, we select the threshold that yields the highest RS(10), which is -3.3 where RS(10) = 12.55. The RS values under different thresholds are summarized in Figure 6, Figure 7 and Figure 8 in Appendix B.

Based on the results obtained from the validation data, we observe a clear difference in generation behavior between the two models. LLaMA-Coder-13B achieves a lower RS(0) but a significantly higher RS(N) compared to Deepseek-Coder-6.7B, indicating that LLaMA-Coder-13B is more conservative in its predictions. Although it produces

fewer correct SQL answers, it also generates fewer incorrect ones.

5.3 Metric

We utilize the RS (Lee et al., 2024b) as our metric. The goal of this metric is to evaluate the reliability of a SQL generation system, where we reward correct SQL generation and penalize incorrect results. The generated SQL is considered correct if the queried result is the same as the ground truth. Hence, the generated SQL can still be accurate as long as the logic of the SQL is similar to the ground truth.

Rewards will be provided if the system generates a correct SQL for answerable questions Q_{ans} or returns 'null' for unanswerable questions Q_{una}. However, A penalty $-c$ will be given if the system gives a wrong SQL for Q_{ans} or returns a SQL for unanswerable questions Q_{una}. No penalty will be given if the system abstains from answering Q_{una} since it is neither helpful nor harmful to the user. The overall RS is calculated by taking the average of sample-level scores, represented in percentages.

Following the EHRSQL 2024 Challenge (Lee et al., 2024c), the primary metric we will be using is RS(10), meaning penalty $c = 10$. The other metric we will be using is RS(0) and RS(N), where N is the number of testing data.

$$\phi_c(x) = \begin{cases} 1 & \text{if } x \in Q_{\text{ans}},\ g(x) = 1,\ \text{Acc}(x) = 1 \\ 0 & \text{if } x \in Q_{\text{ans}},\ g(x) = 0 \\ -c & \text{if } x \in Q_{\text{ans}},\ g(x) = 1,\ \text{Acc}(x) = 0 \\ -c & \text{if } x \in Q_{\text{una}},\ g(x) = 1 \\ 1 & \text{if } x \in Q_{\text{una}},\ g(x) = 0 \end{cases}$$

6 Results

We will compare our result with the other state-of-the-art approach from the EHRSQL 2024 Challenge (Lee et al., 2024c). Based on the results in Table 2, our system outperforms other approaches using open-source LLMs in terms of RS(0), indicating that our method provides more correct answers to users. We also observe a slight improvement in RS(10), attributed to the increased number of correct answers. However, this improvement is limited, as it is accompanied by a rise in incorrectly answered questions. This is reflected in RS(N), where our method falls behind other approaches. This reflects a trade-off: as the number of abstained questions decreases in an effort to provide more correct answers, the likelihood of the system returning incorrect answers increases.

When comparing with systems using proprietary LLMs, our method performs less effectively, highlighting the significant performance gap that still exists between proprietary and open-source models.

7 Ablation Study

7.1 Evaluating Method Effectiveness

To test the effectiveness of our method, we evaluate the system using the testing data on both fine-tuned LLMs, before and after applying the ensemble approach. From Table 3 and Table 4, we observe that LLaMA-coder-13B is more conservative, as it abstains from answering more questions compared to Deepseek-coder-6.7B, resulting in a lower error rate. In contrast, Deepseek-coder-6.7B is more daring and less likely to abstain, which leads to a higher error rate but also enables it to produce more correct answers.

After applying confidence filtering to Deepseek-Coder-6.7B, we observe an increase in both RS(10) and RS(N), indicating a reduction in error rate as low-confidence answers are filtered out. However, this improvement comes at the cost of a decrease in RS(0), bringing its performance closer to that of LLaMA-Coder-13B. We believe the reason we do not observe a clear difference in generation behavior when running the LLMs on the testing data is due to the difference in their capabilities toward certain types of questions. Compared to the result in the validation data, LLaMA-Coder-13B performs better on the testing data, while Deepseek-Coder-6.7B does not perform as well as expected, even though the optimal threshold was selected using the validation data.

With the ensemble method, we observe a significant increase in the number of correct answers, demonstrating the ability of both models to complement each other's strengths. However, the number of incorrect answers also increases, as indicated by a lower RS(N) compared to LLaMA-coder-13B and Deepseek-coder-6.7B with confidence filtering. While RS(0) and RS(10) improve due to the higher number of correct responses, further investigation is needed to reduce the overall error rate of the ensemble system.

	RS(0)	RS(10)	RS(N)	Model Used
LG AI Research & KAIST	**88.17**	**81.32**	**-711.83**	ChatGPT
PromptMind	82.6	74.89	-817.4	GPT-4, ChatGPT, ClaudeOpus
ProbGate	81.92	74.21	-818.08	ChatGPT
Open-source Ensemble (ours)	82.69	44.99	-4317.31	Deepseek-coder-6.7B, LLaMA-coder-13
AIRI NLP	68.89	44.04	-2831.11	T5-3B, Logistic Regression
LTRC-IIITH	66.84	43.7	**-2633.16**	SQLCoder-7B-2
TEAM_optimist	14.14	-713.37	-84.9 K	SQLCoder-7B-2
Open-source Ensemble (ours)	**82.69**	**44.99**	-4317.31	Deepseek-coder-6.7B, LLaMA-coder-13

Table 2: Our results compared with other baselines. The upper section compares our method with approaches that use proprietary LLMs, while the lower section compares it with approaches using open-source LLMs.

Model	RS(0)	RS(10)	RS(N)
Deepseek-coder-6.7B	78.06	15.51	-7221.93
LLaMA-coder-13B	68.04	36.33	-3631.96
Deepseek + confidence filter	68.98	38.99	-3431.02
Ensemble	**82.69**	**44.99**	-4317.31

Table 3: Results before and after introducing the confidence filtering and ensemble method.

7.2 Comparing with Proprietary LLMs

In this experiment, all models use the same zero-shot prompt. Proprietary LLMs without fine-tuning will be given more hints for them to better understand the requirements:

> You are given SQL table schema and the question. Generate the SQL query for the following question. Note that you should generate 'null' if the question cannot be converted to SQL query using the given information. **Your SQL must be executable using Python sqlite.**
> .....

As shown in Table 5, a fine-tuned proprietary LLM can easily outperform our system in terms of reliability and accuracy, where finetuned GPT-4o-mini performs the best with an RS(0) of 93.48 and RS(10) of 68.21. However, fine-tuning such models often raises privacy concerns, particularly regarding patient information, since the process typically cannot be conducted locally. In contrast, our method avoids these concerns entirely, making it more suitable for privacy-sensitive applications. Moreover, it outperforms proprietary LLMs that have not been fine-tuned, which tend to lack sufficient knowledge about the target database.

7.3 Evaluating Both LLM on Different Difficulty

In this experiment, we categorize the test data into three difficulty levels using GPT-4o-mini. Representative SQL examples for each level are provided below.

1. **Easy:** Queries involving basic clauses such as `WHERE`, `ORDER BY`, `LIMIT`, or simple datetime filters.
2. **Medium:** Queries that include joins, basic subqueries, or aggregations (e.g., `COUNT`, `AVG`).
3. **Hard:** Queries containing window functions (e.g., `RANK`, `LAG`), nested subqueries, or complex derived tables.

Table 6 shows the statistic regarding the testing data's difficulty. There are only 75 easy questions in the test set, whereas the numbers of medium and hard questions are 319 and 540, respectively, highlighting the overall difficulty of the dataset.

Based on Table 7, we can see that the finetuned GPT-4o mini is able to perform well on most questions regardless of the difficulty, achieving RS(0) about 90 for every difficulty. For easy difficulty questions, it achieve 90.67 across RS(0) to RS(N), meaning it did not return any wrong results, but was not able to achieve 100% as it still abstain from some questions in the easy category.

Model	Unanswerable correct	Unanswerable wrong	Answerable correct	Answerable abstain	Answerable wrong
Deepseek-Coder-6.7B	215	18	696	183	55
LLaMA-Coder-13B	**231**	**2**	563	336	35
Deepseek + confidence filtering	226	7	579	327	**28**
Open-source Ensemble	225	8	**740**	**158**	36

Table 4: Results details before and after introducing the confidence filtering and ensemble method.

Model	RS(0)	RS(10)	RS(N)
GPT-4.1-mini	45.67	-315.08	-42054.33
GPT-4o-mini	38.22	-259.13	-34661.78
GPT-o4-mini	48.59	-267.61	-36851.41
Gemini-2.5-pro	46.10	-212.68	-30153.90
Gemini-2.5-flash	43.53	-204.97	-28956.47
Open-source Ensemble	82.69	44.99	-4317.31
Finetuned Gemini-2.0-flash	91.00	48.16	-4900.00
Finetuned GPT-4o-mini	**93.48**	**68.21**	**-3206.51**

Table 5: Comparison with proprietary LLMs.

Looking at our finetuned open-source language models, LLaMA-Coder-13B performs well when it comes to questions with easy or medium difficulty, where it achieved an RS(0) of 74.67 and 85.27, respectively. However, when it comes to hard questions, it can only achieve an RS(0) of 43.53 and an RS(10) of -10.19. Deepseek-Coder-6.7B on the other hand, was able to perform almost equally across all difficulty, achieving RS(0) above 60 for all of them.

The statistics highlight a clear difference in performance between the two open-source language models, despite being fine-tuned on the same training data. LLaMA-Coder-13B performs well on easy and medium-difficulty questions but struggles to answer hard questions correctly most of the time. In contrast, while Deepseek-Coder-6.7B could not perform as well when compared to LLaMA-Coder-13B on easy and medium questions, it was able to significantly outperforms it on hard questions.

7.4 Data and Code Availability

The data we will be using is the EHRSQL 2024 dataset by (Lee et al., 2024c), which is available on their website https://sites.google.com/view/ehrsql-2024. Codes and the finetuned model will be available at https://anonymous.4open.science/r/EHR_text_to_sql-AD50/README.md.

Difficulty	Number of testing data
Easy	75
Medium	319
Hard	540

Table 6: Number of questions in different difficulties.

Model	Difficulty	RS(0)	RS(10)	RS(N)
LLaMA-Coder-13B	Easy	74.67	61.33	-1481.33
	Medium	85.27	69.59	-1743.89
	Hard	43.52	-10.19	-6223.70
Deepseek-Coder-6.7B	Easy	68.00	68.00	68.00
	Medium	63.63	47.96	-1765.52
	Hard	60.19	17.59	-4910.37
Finetuned GPT-4o-mini	Easy	90.67	90.67	90.67
	Medium	97.17	87.77	-1000.31
	Hard	91.48	61.85	-3366.29

Table 7: Performance of different LLM models on questions of varying difficulty

References

Ishneet Kaur Chadha, Anshul Gupta, Subham Sarkar, Minakshi Tomer, and Tripti Rathee. 2025. Performance evaluation of open-source llms for text-to-sql conversion in healthcare data. In *2025 International Conference on Pervasive Computational Technologies (ICPCT)*, pages 606–609.

Satya K Gundabathula and Sriram R Kolar. 2024. Promptmind team at ehrsql-2024: Improving reliability of sql generation using ensemble llms.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence.

Yongrae Jo, Seongyun Lee, Minju Seo, Sung Ju Hwang, and Moontae Lee. 2024. Lg ai research kaist at ehrsql 2024: Self-training large language models with pseudo-labeled unanswerable questions for a reliable text-to-sql system on ehrs.

Sangryul Kim, Donghee Han, and Sehyun Kim. 2024. Probgate at ehrsql 2024: Enhancing sql query generation accuracy through probabilistic threshold filtering and error handling.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024a. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation.

Gyubok Lee, Woosog Chay, Seonhee Cho, and Edward Choi. 2024b. Trustsql: Benchmarking text-to-sql reliability with penalty-based scoring.

Gyubok Lee, Sunjun Kweon, Seongsu Bae, and Edward Choi. 2024c. Overview of the ehrsql 2024 shared task on reliable text-to-sql modeling on electronic health records.

Oleg Somov, Alexey Dontsov, and Elena Tutubalina. 2024. AIRI NLP team at EHRSQL 2024 shared task: T5 and logistic regression to the rescue. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*, pages 431–438, Mexico City, Mexico. Association for Computational Linguistics.

Jerrin Thomas, Pruthwik Mishra, Dipti Sharma, and Parameswari Krishnamurthy. 2024. LTRC-IIITH at EHRSQL 2024: Enhancing reliability of text-to-SQL systems through abstention and confidence thresholding. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*, pages 697–702, Mexico City, Mexico. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Riikka Vuokko, Päivi Mäkelä-Bengs, Hannele Hyppönen, Minna Lindqvist, and Persephone Doupi. 2017. Impacts of structuring the electronic health record: Results of a systematic literature review from the perspective of secondary use of patient data. *International journal of medical informatics*, 97:293–303.

A First Appendix

To better understand the differences in generation capabilities between the two finetuned models, we evaluated their performance on the testing data. We analyzed how each model contributes to the overall performance of the ensemble model.

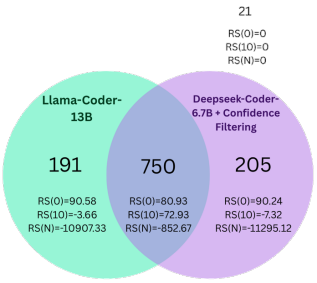

Figure 2: Venn diagram illustrating the complementary generation behaviors of the two fine-tuned models on the test set. The overlapping region represents questions for which both models produce equivalent results, while the non-overlapping regions indicate questions uniquely answered by one model. RS scores are reported for each subset to highlight the contribution of model agreement and disagreement to overall performance.

The results are presented in Figure 2, where the number in the center represents the number of questions answered by both models. According to the figure, 21 questions remain unanswered. These include cases where both models abstain from answering, as well as cases where both models return valid SQL queries but yield different results. Since 'null' will be returned in these cases, RS(0) to RS(N) will always remain 0.

The intersecting region between both models represents cases where both return the same result, covering 750 questions, with an RS(0) of 80.93 and an RS(10) of 72.93. We observe that these cases perform well on the RS(10) metric, highlighting how combining multiple LLMs enables cross-examination of answers, thereby reducing the error rate.

The non-intersecting regions represent cases where only one of the models provides a result. LLaMA-Coder-13B answered 191 questions with an RS(0) of 90.58, while Deepseek-Coder-6.7B handled 205 questions with an RS(0) of 90.28. Although these cases achieve higher RS(0) scores, their RS(10) scores are comparatively lower. In this experiment, both models exhibit negative RS(10) scores in the non-intersecting cases, likely due to the absence of a cross-checking mechanism.

Analyzing the types of queries handled by each model, we observe that LLaMA-Coder-13B performs well with simpler conditions and joins, particularly excelling in datetime-related expressions. Hence, resulting in better performance in easy and medium difficulty questions as shown in Table 7. Deepseek-Coder-6.7B, on the other hand, was able to handle multi-table joins or nested subqueries better than LLaMA-Coder-13B. Although Deepseek-Coder-13B was not able to excel in easy or medium difficulty questions, it was able to maintain its performance in hard difficulty questions, while LLaMA-Coder-13B failed to do so.

B Second Appendix

Below shows the RS for LLaMA-Coder-13B and Deepseek-Coder-6.7B on the validation data using different threshold.

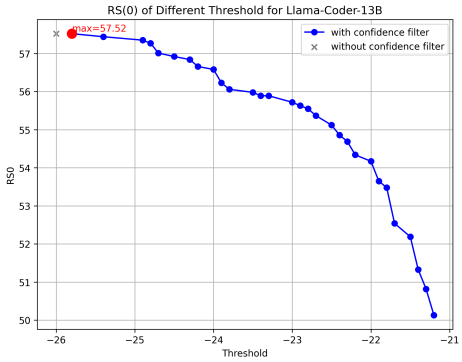

Figure 3: The RS(0) of applying different thresholds on validation data using LLaMA-Coder-13B.

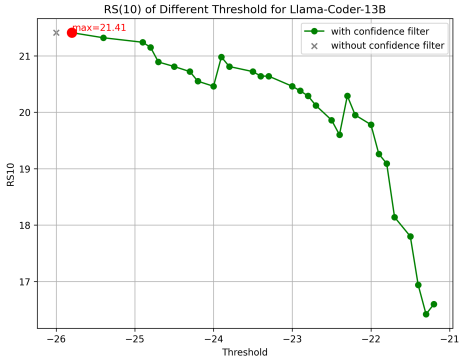

Figure 4: The RS(10) of applying different thresholds on validation data using LLaMA-Coder-13B.

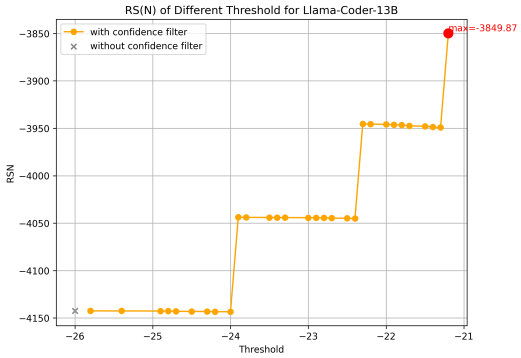

Figure 5: The RS(N) of applying different thresholds on validation data using LLaMA-Coder-13B.

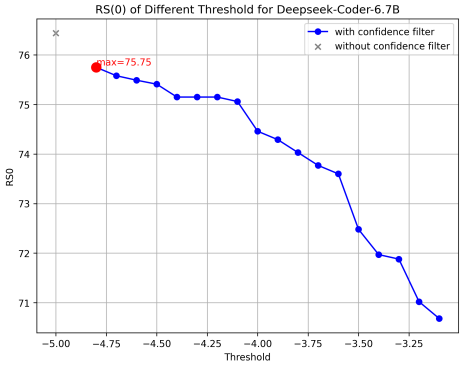

Figure 6: The RS(0) of applying different thresholds on validation data using Deepseek-Coder-6.7B.

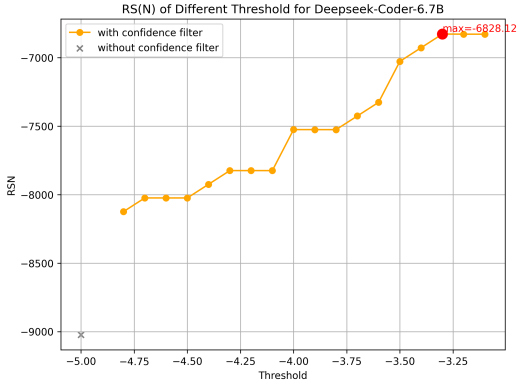

Figure 8: The RS(N) of applying different thresholds on validation data using Deepseek-Coder-6.7B.

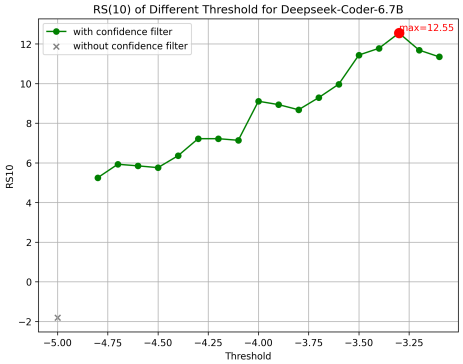

Figure 7: The RS(10) of applying different thresholds on validation data using Deepseek-Coder-6.7B.