# EHRSQL: A practical text-to-SQL benchmark for Electronic Health Records -Final Project-

**Justin Quach**
{jmquach2}@illinois.edu

Group ID: 142
Paper ID: 40
Presentation link: NA
Code link: https://github.com/Justinquach99/EHRSQL

## 1 Introduction

This paper aims to make use of Electronic Health Records (EHRs) to tackle question answering. This will take the overall text from a given list of questions and consequently generate SQL queries. From those queries, we are able to set thresholds to determine whether or not if a prediction confidence exceeds or is below that threshold. If the question's prediction confidence exceeds a given threshold, the resulting SQL query will not be generated and thus return a NULL value. The final output will display values corresponding between zero to one-hundred, one-hundred being the highest and zero being the lowest. These values indicate how well it correctly recognizes as many possible answerable questions (while disregarding unanswerable queries) to gauge the overall model performance.

## 2 Scope of Reproducibility

Aiming for higher accuracy levels (somewhere as close to one-hundred would be perfect) is the goal of this paper. These higher accuracy levels dictate how accurate the generated SQL queries as a result of the question generation process. This process takes a basic, yet vague question and applies specific wording at each stage, thus resulting in a question that is easily interpret-able to become SQL queries. To note, all unanswerable SQL queries are omitted, and only answerable queries are used in evaluating the final result.

### 2.1 Testing Claims For Evaluation

Possible claims for testing:

- Increasing threshold levels (0.50, 0.75, 1, etc.) should decrease the final accuracy values during evaluation (the more SQL queries that are filtered, the less is there to work with).
- Decreasing threshold levels should increase the final accuracy values during evaluation (0.15, 0.10, 0.05, etc.).
- Decreasing total steps during the training process should decrease the final accuracy values during evaluation (since presumably there is less refined data to work with).

## 3 Methodology

For this paper, I aimed to reproduce the paper to the extent that was possible. Using the author's provided EHRSQL GitHub page, I attempted to obtain final results that were similar to the final outputs from their paper. Some aspects of the paper could not be reproduced due to informational limitations (for instance, not having the api-key to run the Codex commands). Instead, I will aim to reproduce the paper by changing other parameters (steps and thresholds for example).

### 3.1 Model descriptions

- Architecture: The authors did not mention a specific architecture that was used for their models. Although the authors' citations mention Question Answering for Complex Electronic Health Records Database using Unified Encoder-Decoder Architecture as a possible architecture utilized, there is insufficient evidence to say with certainty what specific architecture the authors had actually used.
- Learning Objective: The learning objective is question-answering using electronic health data (to be specific, the authors used MIMIC-III and eiCU from the PhysioNet database). Again, questions are passed through a question generation process that becomes incrementally more specific with each passing stage. These questions are then used to gen-

erate SQL queries, and the final resulting output are the accuracy values that dictate how well these queries were generated (on a scale of zero to one-hundred; higher values being deemed more accurate and lower values being deemed less accurate).

- Number of Parameters: Since the parameters are used for training the question data and resulting SQL queries, we can refer to the GitHub page. Under the section 'Question and SQL', train.json contains fifteen parameters. These parameters include the following: db-id, question, template, query, value, q-tag, t-tag, o-tag, tag, department, importance, para-type, is-impossible, split, and id.

## 3.2 Data descriptions

The datasets used were MIMIC-III and eiCU from the PhysioNet database. Accessing both datasets require the user to be credentialed and be granted permission before use. I had been granted credentialed access to these specific databases before attempting this paper.

MIMIC-III database comprises of de-identified health-related data with over forty-thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012. The database provides information like demographic(s), vital sign measurements, laboratory test results, procedures, medications, caregiver notes, imaging reports, and mortality. More information about MIMIC-III database can be found here: MIMIC-III Clinical Database

eiCU database comprises of de-identified health data with over two-hundred-thousand admissions to ICUs across the United States between 2014 and 2015. The database provides information like vital sign measurements, care plan documentation, severity of illness, diagnosis information, and treatment information. More information about eiCU database can be found here: eiCU Clinical Database

## 3.3 Hyperparameters

All relevant hyperparameters were set-up already in the config.py code. For instance, I can change the amount of steps that must be processed during training, the number of total workers in relation to the machine that the code is being ran on, etc..

## 3.4 Implementation

I will make use of existing code courtesy of the author's GitHub page.

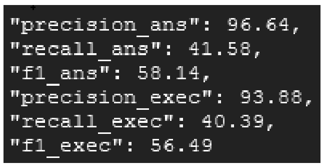
## 3.5 Computational requirements

I made use of Google Cloud's virtual machine to run all the training simulations and evaluations. The machine type is 'e2-standard-16' with 16 vCPU and 64GB memory. The aforementioned base unit described was insufficient in providing the full training process. This was because the training process used more memory than expected and depleted the entirety of said memory before the training reached completion. I raised the total memory size to 500GB (max memory amount that can be changed on a free-trial), which may still not be sufficient to run and store all training files, but this set-up will allow me to fully complete one training cycle (the cycle by default is set to 100000 steps). Currently, I have completed training for both eiCU and MIMIC-III datasets, which in total took over 300GB worth of trained data. I am still, as of this rough draft, currently running training simulations with less steps to gauge overall accuracy performances. The evaluations did not use up any memory space, as the code file's main purpose is to provide statistics like F1-score, precision and recall.

## 4 Results Section

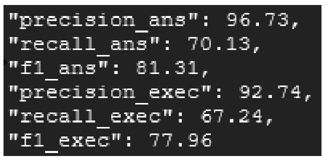
So far, it seems that all of the claims I had made in section 2.1 were supported by the results below. Only some comparisons will be shown to meet page requirement (all comparisons can be found on my GitHub). I will only display some of the final evaluations to save space (the rest of the screenshots can be found on my GitHub page under the COMPLETE prefix folders).

### 4.1 Current Results (MIMIC-III)



```
"precision_ans": 96.64,
"recall_ans": 41.58,
"f1_ans": 58.14,
"precision_exec": 93.88,
"recall_exec": 40.39,
"f1_exec": 56.49
```

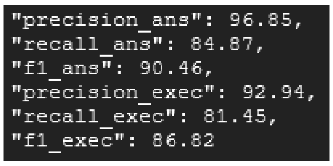
(a) 50000 steps



```
"precision_ans": 96.73,
"recall_ans": 70.13,
"f1_ans": 81.31,
"precision_exec": 92.74,
"recall_exec": 67.24,
"f1_exec": 77.96
```

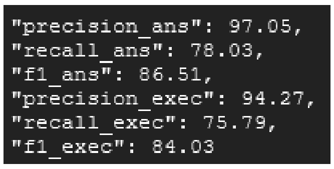
(b) 100000 steps

Figure 1: Threshold value of 0.01



```
"precision_ans": 96.85,
"recall_ans": 84.87,
"f1_ans": 90.46,
"precision_exec": 92.94,
"recall_exec": 81.45,
"f1_exec": 86.82
```

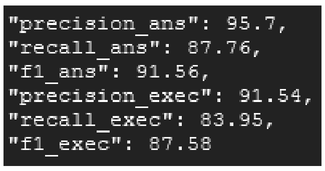
(a) 50000 steps



```
"precision_ans": 97.05,
"recall_ans": 78.03,
"f1_ans": 86.51,
"precision_exec": 94.27,
"recall_exec": 75.79,
"f1_exec": 84.03
```

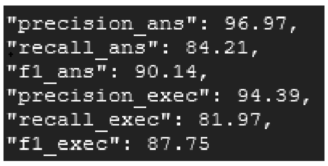
(b) 100000 steps

Figure 2: Threshold value of 0.05



```
"precision_ans": 95.7,
"recall_ans": 87.76,
"f1_ans": 91.56,
"precision_exec": 91.54,
"recall_exec": 83.95,
"f1_exec": 87.58
```

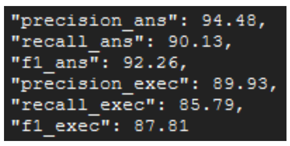
(a) 50000 steps



```
"precision_ans": 96.97,
"recall_ans": 84.21,
"f1_ans": 90.14,
"precision_exec": 94.39,
"recall_exec": 81.97,
"f1_exec": 87.75
```

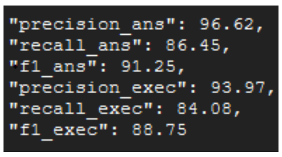
(b) 100000 steps

Figure 3: Threshold value of 0.10



```
"precision_ans": 94.48,
"recall_ans": 90.13,
"f1_ans": 92.26,
"precision_exec": 89.93,
"recall_exec": 85.79,
"f1_exec": 87.81
```

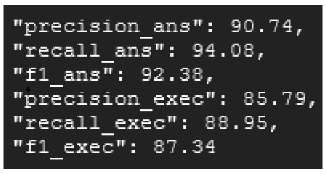
(a) 50000 steps



```
"precision_ans": 96.62,
"recall_ans": 86.45,
"f1_ans": 91.25,
"precision_exec": 93.97,
"recall_exec": 84.08,
"f1_exec": 88.75
```

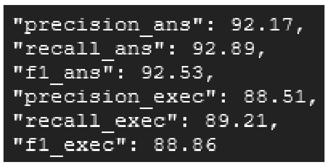
(b) 100000 steps

Figure 4: Threshold value of 0.14144589 (default)



```
"precision_ans": 90.74,
"recall_ans": 94.08,
"f1_ans": 92.38,
"precision_exec": 85.79,
"recall_exec": 88.95,
"f1_exec": 87.34
```

(a) 50000 steps

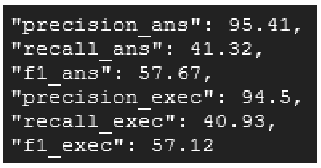


```
"precision_ans": 92.17,
"recall_ans": 92.89,
"f1_ans": 92.53,
"precision_exec": 88.51,
"recall_exec": 89.21,
"f1_exec": 88.86
```

(b) 100000 steps

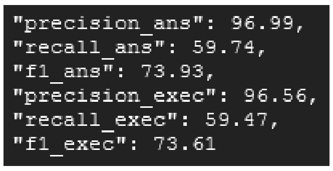
Figure 5: Threshold value of 0.50

### 4.2 Current Results (eiCU)



```
"precision_ans": 95.41,
"recall_ans": 41.32,
"f1_ans": 57.67,
"precision_exec": 94.5,
"recall_exec": 40.93,
"f1_exec": 57.12
```

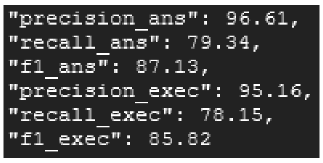
(a) 50000 steps



```
"precision_ans": 96.99,
"recall_ans": 59.74,
"f1_ans": 73.93,
"precision_exec": 96.56,
"recall_exec": 59.47,
"f1_exec": 73.61
```

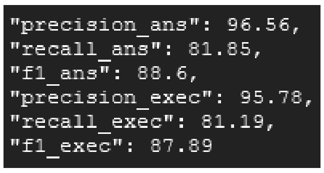
(b) 100000 steps

Figure 6: Threshold value of 0.01



```
"precision_ans": 96.61,
"recall_ans": 79.34,
"f1_ans": 87.13,
"precision_exec": 95.16,
"recall_exec": 78.15,
"f1_exec": 85.82
```

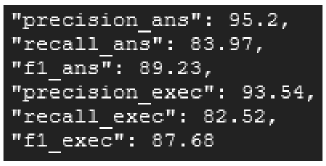
(a) 50000 steps



```
"precision_ans": 96.56,
"recall_ans": 81.85,
"f1_ans": 88.6,
"precision_exec": 95.78,
"recall_exec": 81.19,
"f1_exec": 87.89
```

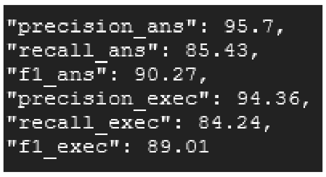
(b) 100000 steps

Figure 7: Threshold value of 0.05



```
"precision_ans": 95.2,
"recall_ans": 83.97,
"f1_ans": 89.23,
"precision_exec": 93.54,
"recall_exec": 82.52,
"f1_exec": 87.68
```

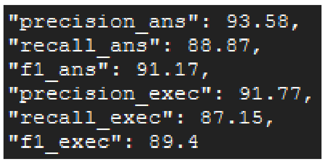
(a) 50000 steps



```
"precision_ans": 95.7,
"recall_ans": 85.43,
"f1_ans": 90.27,
"precision_exec": 94.36,
"recall_exec": 84.24,
"f1_exec": 89.01
```

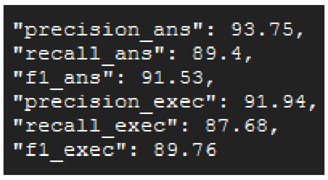
(b) 100000 steps

Figure 8: Threshold value of 0.10



```
"precision_ans": 93.58,
"recall_ans": 88.87,
"f1_ans": 91.17,
"precision_exec": 91.77,
"recall_exec": 87.15,
"f1_exec": 89.4
```

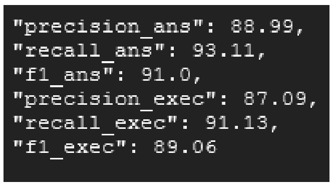
(a) 50000 steps



```
"precision_ans": 93.75,
"recall_ans": 89.4,
"f1_ans": 91.53,
"precision_exec": 91.94,
"recall_exec": 87.68,
"f1_exec": 89.76
```

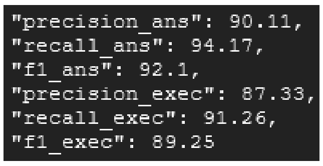
(b) 100000 steps

Figure 9: Threshold value of 0.22580192 (default)



```
"precision_ans": 88.99,
"recall_ans": 93.11,
"f1_ans": 91.0,
"precision_exec": 87.09,
"recall_exec": 91.13,
"f1_exec": 89.06
```

(a) 50000 steps



```
"precision_ans": 90.11,
"recall_ans": 94.17,
"f1_ans": 92.1,
"precision_exec": 87.33,
"recall_exec": 91.26,
"f1_exec": 89.25
```

(b) 100000 steps

Figure 10: Threshold value of 0.50

## 5 Discussion

After performing the full 100000 training cycle, and evaluating the resulting data for both MIMIC-III and eiCU datasets, it seems that increasing the threshold levels does indeed lower the overall accuracy levels. Most likely, this is because, as the threshold levels increase (0.25, 0.50, 0.75, etc.), a greater score means less confidence in the data, which also means that the generated SQL queries should be withheld. So far, the results I had obtained for both MIMIC-III and eiCU were almost similar to what was listed on their paper. This was a good sign, as it took a whole week to run the training simulations for 100000 steps for both datasets.

There are some interesting observations that I noticed when comparing the 50000 and 100000 steps. For instance, aside from 0.01 (applies for both MIMIC-III and eiCU), the recall-ans was significantly higher for 50000 than 100000 steps for eiCU (applies for 0.05, 0.10, 0.14144589, 0.50). However, for MIMIC-III, aside from 0.01, the recall-ans was significantly lower for 50000 than 100000 steps. The F1-ans also follows a similar trend where the values decreased for 100000 steps than 50000 steps for MIMIC-III and the values increased for eiCU.

Another observation that I noticed would be that, aside from threshold value 0.01, most of the values for precision-ans, recall-ans, f1-ans, precision-exec, recall-exec, and f1-exec were relatively high (being over an arbitrary value of 0.80), with some insignificant outliers. However, as soon as the threshold value exceeds 1, the overall values tend to see a decrease in confidence. Most likely, the codes have some difficulty performing evaluations for values not being between 0 to 1 (if we read this as percentage-based, 0 being 0 percent and 1 being 100 percent). The screenshots provided on my GitHub does support this observation.

All of these observations provide useful information when it comes to figuring out what the user (the one who will eventually run and compile these files) wants from extracting electronic health records data into readable SQL queries. These threshold values provide useful evaluation metrics that allow users to view and see which threshold value would approximately give the most optimal results, which then the user can apply for their given situation. There is a significant difference between 50000 and 100000 steps; there is more confidence in the trained data for 100000 steps since there is more data to train and evaluate, and 50000 steps is quicker to train and has less confidence than 100000 steps. Despite this, the final evaluation values (excluding 0.01) are relatively close to one another despite the large differences in the total amount of steps.

### 5.1 What was Easy

Once everything has been setup (all files in one folder, commands ready to use, all necessary libraries installed), the training process and producing different threshold levels was quite a simple task. The training process can be left in the background thanks to the nohup prefix appended to the command. Nohup essentially runs the code in the background, and even if the terminal terminates (like a timeout), the process will not be interrupted (this is also assuming the training process has enough memory to be left in the background without using the entire memory space). The threshold levels can be changed at will.

### 5.2 What was Difficult

During the early stage before training, I utilized the author's GitHub at its base (without modifying any of the files). For some reason, I could not run any of the commands without stumbling onto directory pathing issues. Even if the directory pathing can be found within the same folder as the files (main.py, etc.), the commands simply could not find these external files, even when they actually exist. To counter this issue, I simply moved all the files and modified the directory pathing for all files so that it suits my needs so that I would not have any more pathing issues.

Another thing that was difficult was understanding how to use Google Cloud's service. The UI was frustrating to use, and looking up resources on navigating these issues were met with outdated and/or insufficient guides. In addition, the base virtual machine (16GB) provided by Google Cloud proved insufficient, as the training process stopped at approximately 10000-15000 steps out of 100000. I raised the memory space to 64GB, but the training stopped at approximately 35000. Finally, I simply changed it to the maximum allotted space that I could use on a free trial, which was 500 GB.

Another issue I had impacted the results section. I do not have access to the authors' code for K-means. This means that I do not have a way to know what value was set for the threshold, since the authors never mention what specific value they used. Thus, I cannot output results for K-means to compare with the authors'. The same applies for the schema + T5 section, since I also do not have access to the API key so I could run the codes using their codex.

One other minor issue was that the author's GitHub only listed some of the many libraries that were required to run the code.

### 5.3 Recommendations for Reproducibility

I would recommend the following steps for reproducibility:

1. Ensure that the system can even process and run the entire training step (100000 steps). Every 50 steps, at least when using the Google Cloud specs listed above, takes roughly 5 minutes. In addition, the training process in its entirety can take almost 100 GB worth of memory. The base amount of memory provided by Google Cloud (64 GB) was insufficient in performing training

2. I personally moved all the necessary files (config.py, main.py, etc.) into one folder, along with changing each file's directory pathing so that it suits this particular need. The commands text file will be useful in navigating through this particular setup

3. Ensure that the terminal is equipped with pip, so that one can run something like pip install [library]. The library gsutil is optional and was not impactful for this project. There are other libraries that I may have missed during the installation process

## 6 Communication with Original Authors

I had communicated with the original authors to clarify on two questions I had prior to starting the project:

1. I wanted to know how to obtain the prediction-raw.json, or the prediction.json files. I assume this is obtained after finishing the T5 SQL generation training fully (after 100,000 steps have been performed), but I simply wanted to confirm if what I said above is correct. If this is not the case, where can I obtain the prediction-raw.json or the prediction.json files, since these files are not listed on the GitHub page?

2. How do the threshold values in this command listed on the GitHub page affect the output? Do these threshold values act as a filter of what SQL queries get generated (for instance, some SQL queries will be generated NULL as output, or some other probable solution)?

I received an answer a day after I had sent the authors my inquiries. Their answers were concise and helped to confirm the answers I had originally.

## References

[1] Johnson, A., Pollard, T., Mark, R., *MIMIC-III Clinical Database v1.4*, https://physionet.org/content/mimiciii/1.4/, 2016, September 4.

[2] Pollard, T., Johnson, A., Raffa, J., Celi, L. A., Badawi, O., Mark, R. *eICU Collaborative Research Database v2.0*, https://physionet.org/content/eicu-crd/2.0/, 2019, April 15.

[3] glee4810, *EHRSQL: A practical text-to-SQL benchmark for Electronic Health Records*, https://github.com/glee4810/EHRSQL.

[4] Bae, S., Kim, D., Kim, J., Choi, E., *Question answering for complex electronic health records database using unified encoder-decoder architecture*, https://proceedings.mlr.press/v158/bae21a/bae21a.pdf, 2021, November 28.

[5] Lee, G., Hwang, H., Bae, S., Kwon, Y., Shin, W., Yang, S., Seo, M., Kim, J.-Y., Choi, E., *EHRSQL: A Practical Text-to-SQL Benchmark for Electronic Health Records*, https://arxiv.org/abs/2301.07695, 2023, 16 January.