EHR-SeqSQL : A Sequential Text-to-SQL Dataset For Interactively Exploring Electronic Health Records

Anonymous ACL submission

Abstract

Text-to-SQL parsing allows non-expert users to obtain information from databases using natural language. There are several important yet under-explored objectives in this field: interactivity, compositionality, and efficiency. In this paper, we present EHR-SeqSQL, a sequential text-to-SQL dataset, specifically designed for Electronic Health Record (EHR) databases. We demonstrate the benefits of multi-turn setting over single-turn setting with respect to compositionality, and provide a new data split and an additional test set to evaluate compositional generalization. Furthermore, we introduce unique special tokens in SQL queries to enhance execution efficiency. By addressing all these objectives above, our research aims to bridge the gap between industry needs and academic research in the text-to-SQL domain.

1 Introduction

Text-to-SQL provides a practical opportunity for non-experts to explore databases, even without prior knowledge of the database operations. Electronic Health Records (EHRs) are large-scale relational databases (RDBs) storing vast and comprehensive patient data (Johnson et al., 2016; Pollard et al., 2018). Medical experts often ask questions that require highly complex reasoning across multiple tables and access to a vast number of records within a single query (Lee et al., 2022). Handling such complexity in large-scale databases remains a significant challenge in the current text-to-SQL research (Li et al., 2023).

Several efforts have been made to construct text-to-SQL datasets for EHRs. MIMIC-SQL (Wang et al., 2020) is the first text-to-SQL dataset that targets a subset of MIMIC-III (Johnson et al., 2016), one of the widely-used open-source EHR databases, consisting of 26 tables. DrugEHRQA (Bardhan et al., 2022) provides the first question answering dataset that incorporates both structured tables and unstructured notes from EHRs. EHRSQL (Lee et al., 2022) is a dataset curated based on a survey from various medical experts, reflecting the diverse information needs of the actual medical field.

Still, there are important yet under-explored objectives for the practical application of text-to-SQL

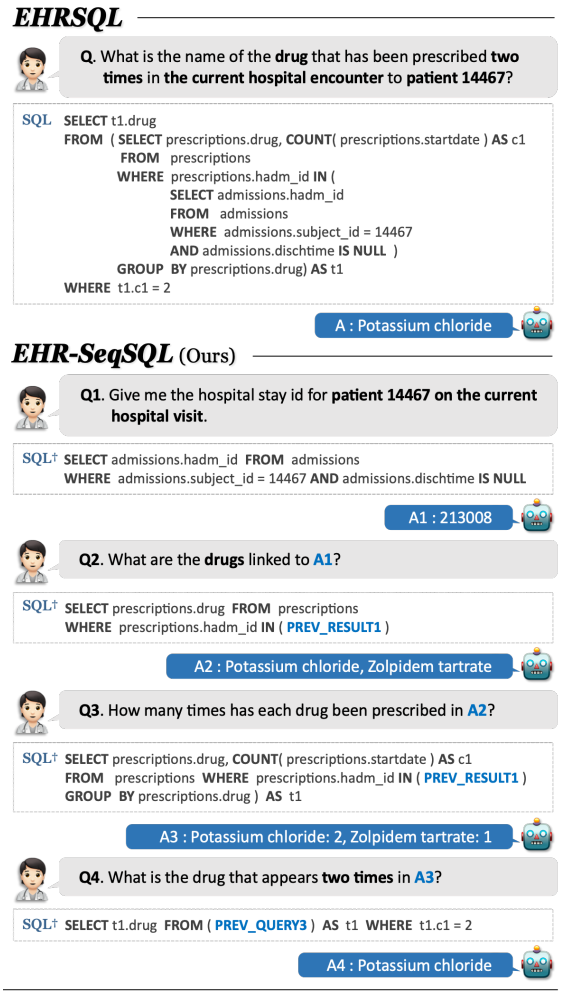

Figure 1: **EHRSQL vs. EHR-SeqSQL** EHR-SeqSQL is a dataset that adapts the single-turn setting of EHRSQL into a multi-turn setting. The SQL queries in EHR-SeqSQL include the special tokens to refer to the previous context, which can be executed in the database with simple post-processing.

models. **First, the text-to-SQL system should incorporate interactivity.** Most of the existing text-to-SQL research assumes a single-turn scenario, whereas the process of exploring information in real-world situations is often continuous (Iyyer et al., 2017; Yu et al., 2019a,b). For better user usability, it is desirable for information retrieval solutions to be interactive as well. **Second, the text-to-SQL system should learn compositionality.** Realistically, datasets cannot capture all the varied needs of users. Therefore, it is crucial that the model can handle a wider range of queries especially when these unseen queries comprise components (*i.e.* sub-queries) that the model has explicitly seen during training. This is the compositional generalization ability, which is quite challenging even for the highly proficient language models (Qiu et al., 2022). **Lastly, the text-to-SQL system should embrace efficiency.** Real-world databases are often significantly larger than academic ones (Hazoom et al., 2021). Efficient query execution is crucial, given the magnitude of the databases in actual hospitals (*e.g.* the original MIMIC-III dataset includes the medical history of 46k patients). This becomes even more significant when we consider a real-time interaction scenario between a text-to-SQL model and a user (Li et al., 2023).

In this work, we introduce EHR-SeqSQL which addresses all three objectives. Our contribution is as follows:

- **Construction of a sequential text-to-SQL dataset for exploring EHR data:** As illustrated in Figure 1, EHR-SeqSQL is designed for multi-step interactions, built by decomposing the diverse and complex queries of EHRSQL and setting each question as the interaction goal. To the best of our knowledge, EHR-SeqSQL is the first multi-turn text-to-SQL dataset targeting structured medical records. We make our dataset public to encourage future research.

- **Validating the effectiveness of EHR-SeqSQL in the compositional generalization:** We designed two experiments to evaluate the compositional generalization ability. Through our experiments, it was found that decomposing questions enables the model to better generalize to unseen interaction goals during training. Furthermore, it demonstrated the potential for generalization on longer sequences not encountered during training.

- **Proposing special tokens for SQL:** We introduce a highly effective way to enhance query execution efficiency in multi-step interactions with the use of the novel special tokens for SQL. We observed that these tokens significantly increase time efficiency during query execution as well as improve the performance of text-to-SQL models. Moreover, the efficiency grows significantly as the database size increases.

2 Related Work

2.1 Multi-turn Text-to-SQL

There are only a few datasets that are specifically designed for context-dependent text-to-SQL tasks. ATIS (Hemphill et al., 1990) was the first to include a series of user questions aimed at interacting with a flight database. Recently, Yu et al. (2019a,b) proposed cross-domain context-dependent text-to-SQL datasets, based on the questions from Spider (Yu et al., 2018) serving as interaction goals. While their motivation aligns closely with ours, their questions and the grounding databases are predominantly simplistic. In real-world scenarios, however, users typically have much more complex requirements, and databases often contain a significantly larger number of rows (Lee et al., 2022; Li et al., 2023). EHR-SeqSQL is designed upon this objective, based on the questions collected from an actual survey and grounding the real-world database, MIMIC-III (Johnson et al., 2016). To the best of our knowledge, this is the first multi-turn text-to-SQL dataset in the healthcare domain.

2.2 Compositional Generalization

Compositional generalization, a major challenge in semantic parsing, enables a model to manage diverse and unfamiliar user queries using components recognized from training. It is typically achieved by creating different training and test splits of interest. Lake and Baroni (2018) discovered that sequence-to-sequence models struggle to learn compositional structures, as demonstrated through their primitive and length splits. More recently, template splits (Finegan-Dollak et al., 2018) and TMCD splits (Shaw et al., 2020) were proposed to evaluate compositional generalization on text-to-SQL datasets. In this work, we further extend the concept of compositional generalization into an interactive multi-turn setting within the text-to-SQL domain.

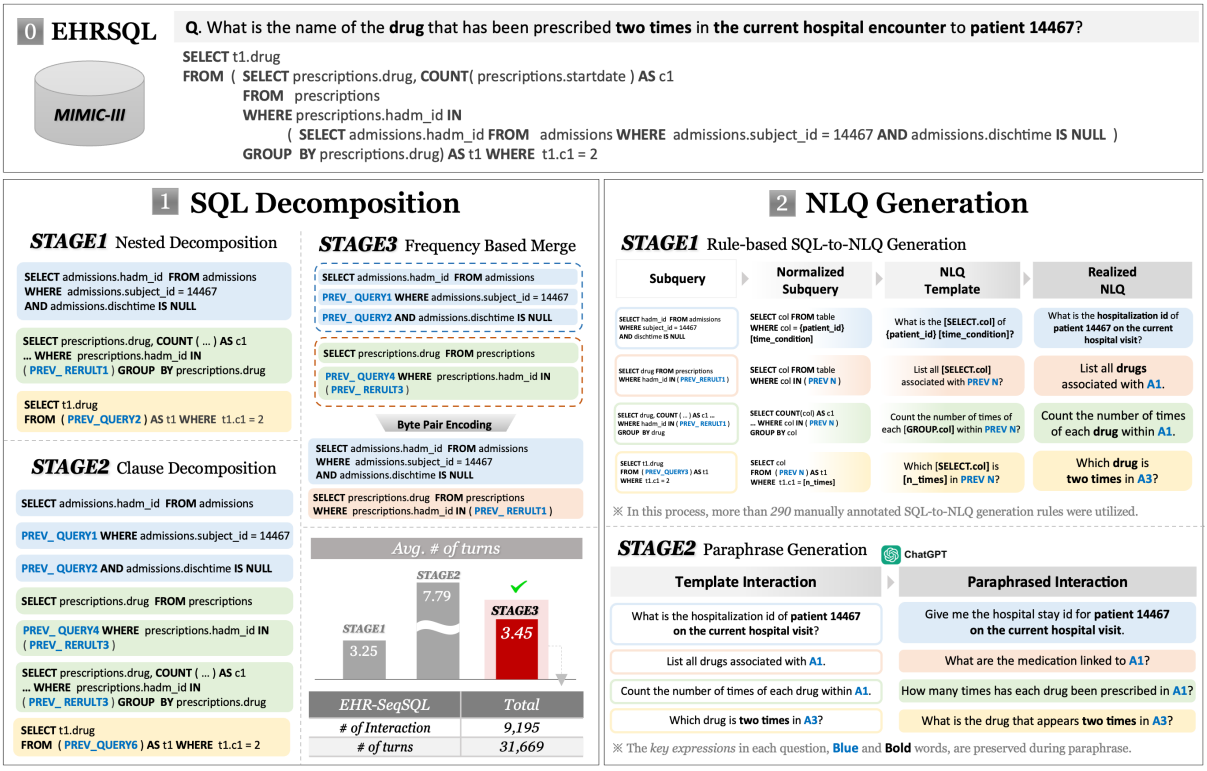

Figure 2: Overview of the dataset construction process. We transform a single text-SQL pair from EHRSQL into multi-turn text-SQL pairs for EHR-SeqSQL. We first decompose the original SQL query into subqueries in Stage 1 and 2, then apply the BPE algorithm to merge frequent subquery patterns in Stage 3. We then generate natural language questions (NLQ) for each subquery based on manually created NLQ templates. Lastly, NLQs are paraphrased using ChatGPT to create the final dataset.

2.3 SQL Efficiency

In domains that utilize large and complex databases such as EHRs, the efficiency of SQL becomes increasingly critical. The majority of existing research focuses on improving SQL efficiency through post-hoc query optimization (Zhou et al., 2021; Li et al., 2023) or efficient database indexing (Zhou, 2022). In contrast to these approaches, we propose special tokens for SQL that specifically target SQL efficiency in a multi-turn setting. Our approach not only improves execution efficiency but also enhances performance, which is further discussed in Section 5.3.

3 Data Construction

We aim to convert text-SQL pairs of EHRSQL to multi-turn text-SQL pairs that embrace interactivity and query execution efficiency. Note that EHRSQL contains text-SQL samples for two different EHR sources, namely MIMIC-III (Johnson et al., 2016) and eICU (Pollard et al., 2018). We use the MIMIC-III version of EHRSQL, given its complex schema and wider adoption by the NLP community.

Our data construction process consists of two parts: SQL decomposition and natural language question (NLQ) generation. In SQL decomposition, we first break down each SQL query of EHRSQL into a sequence of subqueries to create more granular meanings of queries. Next, we create a corresponding NLQ for each subquery. Compared to NLQ decomposition, SQL decomposition enables systematic breakdown of a question without any loss of information, due to the strict, rule-based nature of SQL syntax. The overall process is shown in Figure 2 and we elaborate each step in the following sections.

3.1 SQL Decomposition

Stage 1. Decomposing Nested Query As SQL queries in EHRSQL follow a nested structure, we decompose the EHRSQL queries using their nested structure. In this way, subqueries can contain more granular meanings of the original query, which in turn allows us to create multiple turns of interaction. The inner query is asked first, and then a question corresponding to the outer query is asked while referring to the result of the inner query. This

	ATIS	SParC	CoSQL	**EHR-SeqSQL**
# of Interactions	1,658	4,298	3,007	9,195
# of Turns	11,653	12,726	15,598	31,669
Avg. # of Turns	7.0	3.0	5.2	3.5
Avg. # of Tables / DB	27	5.1	5.1	26
Domain	Airline	Cross	Cross	Medical
Compositional Split	✗	✗	✗	✓
SQL Execution Efficiency	✗	✗	✗	✓

Table 1: Comparison of EHR-SeqSQL with other multi-turn text-to-SQL datasets.

decomposition approach guarantees the executability of queries at each turn and allows anaphoric expressions in the subsequent turns.

Meanwhile, we devised two special tokens for SQL to leverage the previous turn's subqueries and results: `prev_query` and `prev_result`. These tokens are accompanied by a specific turn index (e.g. `prev_query1`, `prev_result2`). Specifically, `prev_query` token refers to the generated query of the specified turn, whereas the `prev_result` token refers to the execution result of the specified turn. Two tokens are substituted with either the referred query or its execution result before execution. The effectiveness of these tokens is later discussed in Section 5.3.

Stage 2. Decomposing SQL Clauses After Stage 1, multi-turn questions are obtained, where each can be asked with reference to the answer to previous questions. Still, these questions often convey multiple conditions that users might ask in separate questions. For example, the question "*What was the last hospitalized admission time that patient 17694 was admitted via transfer from emergency room?*" contains the specific conditions about the admission time, the patient, and the admission route. Therefore, we further decompose queries on the clause-level: WHERE, ORDER BY, and HAVING clauses and aggregation functions (*e.g.* `MAX`, `MIN`, `SUM`) in the SELECT clause. In cases where multiple clauses appear together, we parse in the order of SQL execution. Exceptions were made only when the original semantics of the query became ambiguous after being decomposed. Please see further details in Appendix A.1.

Stage 3. Merging Subqueries by Frequency We noted that frequently appearing consecutive subqueries are most likely decomposed due to the specific database schema characteristics or SQL structures rather than each being queries users would naturally ask. In Stage 3, we employ a recursive application of the Byte Pair Encoding (BPE) algorithm to merge frequent consecutive subquery pairs. This recursive approach allows for the amalgamation of not only two but up to three or more subqueries, thereby simplifying complex patterns and enhancing query interpretability. Further details are given in Appendix A.2.

3.2 NLQ Generation

Stage 1. Rule-based SQL-to-NLQ Generation While previous SQL-to-NLQ studies (Gan et al., 2022; Wu et al., 2021) decomposed SQL at the clause level for sub-questions and concatenated clauses later, such simple concatenation has a risk of unnaturalness. Thus we create NLQ templates for each corresponding SQL subqueries for the quality of NLQ. To efficiently annotate NLQs for each corresponding SQL subqueries, we first normalize subqueries by replacing specific table names, column names, and condition values to abstract terms such as `table` and `col` (see Figure 2). We then manually create NLQ templates for each normalized subqueries. These templates include slots for table names, column names, condition values, SQL functions (*e.g.* GROUP BY, AVG), and time expressions. Then the actual question is generated from the NLQ templates by the slot-filling process. More details are in Appendix A.3.

Stage 2. Paraphrase Generation We further paraphrase the NLQ templates to enhance linguistic variability within the dataset, using ChatGPT for its superior performance in understanding and generating text (Guo et al., 2023). We ensured that all the slot values and turn indices were preserved after paraphrasing. Additionally, we employed the self-consistency method and a duplicate question detection model to ensure the quality of the paraphrases, following Lee et al. (2022). On average, we obtained 10.39 paraphrases for each NLQ tem-

plate. Finally, we randomly assign the paraphrased template to each question and fill the slots with the original condition values.

Quality Check In the process of decomposing SQL queries, we strictly adhered to traditional SQL syntax to ensure the intactness. However, generating NLQs based on templates necessitate meticulous quality checks. On the turn level, we examined the consistency of NLQ templates of each question. Our primary concerns were 1) *Completeness*: We make sure that all information in the SQL is also explicitly stated in the NLQ, and 2) *Naturalness*: We ensured the question templates were as natural as possible after the masked information is realized. On the interaction level, we evaluated whether the interaction accurately represents the original intent of the EHRSQL question. We also carefully analyzed how turns are connected naturally within an interaction through coreference. We performed the turn-level check for every question template and both turn-level and interaction-level quality checks for 1,000 sampled interactions from the final data. When unnatural template or interaction flow is found, we created additional templates or edited the template to be clear until all the sampled instances conformed to our quality standards in both turn and interaction level.

3.3 Data Statistics

Table 1 presents the statistics of EHR-SeqSQL compared to other multi-turn text-to-SQL datasets. EHR-SeqSQL has the largest number of interactions and turns compared to all existing multi-turn text-to-SQL datasets. Notably, this is the first work to introduce special tokens within SQL queries to improve execution efficiency. Furthermore, we also provide a new split and additional test set to evaluate the compositional generalization, as detailed in Section 5.

Questions in our dataset are categorized into four types: *independent*, *referential*, *filtering*, and *modifying*. A question can belong to multiple categories as they are not mutually exclusive. *Independent* questions lack prior context. *Referential* questions refer to previous questions or answers while *filtering* questions narrow the previous question's scope by adding conditions. *Modifying* questions are particularly challenging as they only mention the altered condition and omit all the same conditions. Table 6 shows the distribution of the questions.

4 Experimental Setup

4.1 Task

The objective of the model is to generate a SQL query given the interaction history and current question. We experiment with two versions of interaction history: one that includes only the previous questions, denoted as *QQ*, and another that includes both the previous questions and their corresponding SQL queries, denoted as *QS*. For the *QS* setting, the model is trained with the interaction history using the ground-truth queries, while during the inference phase, the model's own predictions are used in order to simulate a real-world application.

4.2 Baselines

4.2.1 Fine-tuning Models

We employ both a fine-tuning approach and an in-context learning approach for our baselines. For the fine-tuning approach, we use the T5 models (Raffel et al., 2020), the general-purpose sequence-to-sequence models as our baseline models. We did not use the state-of-the-art models for SParC (Yu et al., 2019b) or CoSQL (Yu et al., 2019a) due to their SQL grammar being confined to Spider which is not compatible with our dataset.

4.2.2 In-context Learning Models

We use ChatGPT for our in-context learning models. Instead of a zero-shot approach, we employ few-shot prompting. We retrieve similar exemplars for each test instance using the BM25 algorithm to use as prompt. For every experiment with in-context learning setting, we use 20-shot. For EHRSQL, which is single-turn, computing the similarity for each question is sufficient. However, for EHR-SeqSQL, which is multi-turn and context-dependent, it's necessary to consider both the current question and the interaction history. Therefore, we use 10 retrieved examples related to the current question and another 10 related to the entire interaction history. More details such as prompt and few-shot retrieval in our in-context learning baseline are in Appendix E.

4.3 Evaluation

The two commonly used metrics in text-to-SQL are Exact Match Accuracy (EM) and Execution Accuracy (EX). However, the EM metric can sometimes be overly strict because it doesn't take into account predictions that have different SQL syntax

but yield the same execution result as the ground-truth query. Thus we use the EX score to measure the model performance based on query execution results. Before execution, the special tokens are replaced with the generated queries or their execution result through post-processing, thus any errors from the referenced turn will be propagated.

Additionally, for the multi-turn setting, we adopt Interaction Match (IM) and Question Match (QM) following Yu et al. (2019b) while computing EX. IM measures the accuracy of the entire interaction while QM measures the accuracy of each turn. We utilize appropriate metrics regarding the purpose of each experiment.

5 Experiments

We now empirically demonstrate the benefits of EHR-SeqSQL in terms of two types of compositional generalization (§ 5.1, § 5.2), as well as the effectiveness of the special tokens (§ 5.3).

5.1 Generalization to Unseen Interaction Goals

Compositional Split We first aim to evaluate whether training models in a multi-turn setting can lead to the acquisition of compositional generalization abilities. To explore this aspect, we split our dataset in a compositional manner, namely *compositional split*. This split differs from the *random split* in EHRSQL where the distributions of SQL structures in the training and test sets are nearly identical. In contrast, *compositional split* includes unseen SQL structures in the test set, though these structures can be decomposed into smaller parts that are all present in the training data.

More concretely, we first define the terms *compositions* and *components* and use the concepts to automatically split the dataset. *Compositions* represent SQL templates in EHRSQL, where condition values, SQL functions (i.e. `SUM`, `AVG`, etc), time expressions (i.e. subqueries constraining *last year*, *in 2023*, etc) are masked. *Components* refer to the decomposed SQL template clauses derived from Stage 2, as explained in Section 3.1. Each composition, which corresponds to an interaction goal, contains a set of components that exist as individual SQL subqueries in the dataset. Table 11 provides concrete examples of components and compositions. We employ a greedy algorithm to split the dataset, similar to Shaw et al. (2020). Table 7 provides the statistics of the random and compositional splits. More details are given in Appendix B.

Metric We only measure IM to compare the performance of a model trained on EHRSQL and a model trained on EHR-SeqSQL. Given that an interaction in EHR-SeqSQL corresponds to a question in EHRSQL, correctly predicting every question within an interaction is equal to correctly predicting a single question in EHRSQL.

Result The experimental results are shown in Table 2. In a random split, all models exhibit strong performance with both EHRSQL and EHR-SeqSQL. However, in the compositional split, models typically show superior generalization ability when trained with EHR-SeqSQL. T5 models, in particular, show a significant performance increase, ranging from 7.63%p (from 67.8 to 75.43), to 30.34%p (from 52.15 to 82.49). T5-base finetuned in *QQ* setting has the best generalization ability. It suggests that a task-specific fine-tuning enables effective extraction of necessary information only from NLQs without being distracted by other contextual factors. On the other hand, ChatGPT suffers a performance drop in the *QQ* setting, which is suspected to be due to the absence of predicted SQL queries within the prompt, which leads ChatGPT to have less information on target representation. Still, a multi-turn setting leads to a performance increase of 7.63% in the *QS* setting. It's worth noting that ChatGPT has better compositional generalization ability in a single-turn setting than fine-tuned models. Overall, our experiments show that training with EHR-SeqSQL allows the models to generalize well even with unseen questions.

5.2 Generalization to Longer Interactions

Longer Interaction Generation In this section, we aim to explore whether the models trained with EHR-SeqSQL can comprehend longer interactions, by achieving another type of compositional generalization ability. To test this, we created a new test set ($Test_L$) composed of multiple related interactions that simulate follow-up questions. This mirrors real-world situations where multiple related questions naturally arise as a user often seeks to familiarize themselves with the knowledge they are curious about (Yu et al., 2019b). For each test set of *random split* and *compositional split*, we created 100 longer interaction instances by connecting related interactions, denoting them as $Test_{LR}$ and $Test_{LC}$. The statistics for $Test_L$ can be found in Table 7. These interactions are, on average, five

Model	Random			Compositional		
	EHRSQL	EHR-SeqSQL		EHRSQL	EHR-SeqSQL	
		QQ	*QS*		*QQ*	*QS*
T5-Base	$94.25_{\pm 0.54}$	$94.69_{\pm 0.01}$	$95.38_{\pm 1.48}$	$52.15_{\pm 0.85}$	$82.49_{\pm 1.80}$	$75.38_{\pm 1.94}$
T5-3B	$90.68_{\pm 9.47}$	$93.07_{\pm 3.71}$	$92.38_{\pm 4.9}$	$54.90_{\pm 3.51}$	$71.39_{\pm 1.05}$	$70.07_{\pm 1.85}$
ChatGPT	91.22	80.74	91.37	67.80	60.50	75.43

Table 2: Model performances in two different splits. For the fine-tuning models, we trained each model with three different random seeds. We report the average score and the standard deviation.

Model		Test_{LR}		Test_{LC}	
		QM↑	IFF↑	QM↑	IFF↑
T5-Base	*QQ*	54.13	5.70 (15.17)	50.58	5.08 (16.64)
	QS	71.33	4.51 (8.03)	61.58	3.72 (8.31)
ChatGPT	*QQ*	92.94	11.24 (15.17)	73.66	5.98 (16.64)
	QS	**94.57**	**12.21** (15.17)	**79.67**	**6.81** (16.64)

Table 3: Model performances on longer interactions. For the IFF score, the perfect score for each setting is denoted in parentheses.

times longer than those in the training data. Notably, $\textit{Test}_{LR}$ poses a challenge with regard to generalization for longer sequences, while $\textit{Test}_{LC}$ additionally provides a challenge in generalization for unseen questions, resulting in a more challenging but more practical scenario. Details about the longer interaction generation process can be found in Appendix C.1.

Metric In this experiment, as can be seen in Table 3, we investigate whether a model trained on an average of approximately *three turns* can generalize to interactions of more than an average of 14 turns. Therefore, IM metric is deemed excessively strict. Instead, we report QM to measure the ratio of correctly answered turns. We further measure the Index of the First Failure (IFF), to capture where the model first generates an incorrect response. The methodology for calculating the IFF score is detailed in Appendix C.2.

Result The experiment results are presented in Table 3. Similarly with the findings in Section 5.1, scores from $\textit{Test}_{LR}$ outperform that from $\textit{Test}_{LC}$. This is likely because the model should generalize to the unseen interaction goals as well as longer interactions in $\textit{Test}_{LC}$. Interestingly, IFF scores from all models are higher than the average length of interactions in the training data. This suggests that the models can learn interaction-level compositionality and generalize to longer turns after being trained with EHR-SeqSQL. However, for the *QS* setting, T5-base model had a disadvantage in taking long interactions due to its input length constraint, which led to the lowest IFF score. On the other hand, ChatGPT outperformed the fine-tuned T5 models in both QM and IFF scores, highlighting its adaptability to longer interactions. Especially, in the random split, ChatGPT achieved an IFF score of 12.21 in the *QS* setting, quite close to the ideal score of 15.17.

5.3 Effects of Special Tokens for SQL

In this section, we evaluate the effect of two special tokens–`prev_query` and `prev_result`. These two tokens allow models to easily reference either the previous query or its execution result and thereby alleviate the decoding burden. Additionally, `prev_result` can further reduce the execution overhead by preventing duplicated subqueries to be executed multiple times. We assess the utility of these tokens from two aspects: model performance and query execution efficiency.

5.3.1 Effects on Model Performance

Details First, we evaluate the impact of the special tokens on model performance. We compare a model trained with queries with the original EHR-SeqSQL, which include the special tokens, to another trained with its standard SQL query version. We ensure that all other training factors, such as model architecture, learning rate, or optimizer, remain consistent between the two models. To prevent the test set from being too simple, which could potentially undermine the impact of the special tokens, we use a compositional split. We report both QM and IM to assess the performance of the models at both the question and the interaction level.

Model		SQL		SQL†	
		QM↑	IM↑	QM↑	IM↑
T5-Base	*QQ*	75.37	52.49	**89.97**	**82.49**
	QS	82.10	64.35	**86.46**	**75.38**
ChatGPT	*QQ*	72.07	45.14	**75.15**	**60.50**
	QS	88.22	72.55	**88.23**	**75.43**

Table 4: Comparison of model performance trained with the original SQL queries and queries with our special tokens (SQL†).

Patient	Range	SQL		SQL†	
		avg.↓	med.↓	avg.↓	med.↓
1k	Full	0.22	0.01	**0.18**	**0.01**
	ST	0.20	0.09	**0.09**	**0.01**
10k	Full	9.01	0.05	**7.96**	**0.03**
	ST	3.25	1.17	**1.15**	**0.15**
46k	Full	164.69	0.19	**139.44**	**0.08**
	ST	31.57	4.60	**6.00**	**0.52**

Table 5: Execution time of SQL queries measured in databases of different sizes. We report the statistics for the entire queries (Full) and for the queries that contain the special token, `prev_result` (ST). Units are deciseconds (10^{-1}).

Result As shown in Table 4, the incorporation of special tokens consistently enhances the performance across all settings, regardless of the model variant or the type of interaction history. In the *QQ* setting, where the absence of prior query history makes the contextual questions more challenging, the special tokens contribute to a substantial performance increase. This is because those tokens simplify referencing previous questions. Specifically, at the question level, the use of these special tokens leads to a performance increase of 14.6%p (from 75.37 to 89.97), and 30%p (from 52.49 to 82.49) at the interaction level. In the QS setting, on the other hand, the reference is more straightforward because the previous SQL information is given. Still, special tokens enhance model performance by reducing the complexity of the target representation. ChatGPT demonstrates a robust performance even without the special tokens in the *QS* setting. We speculate this is because ChatGPT is trained on diverse data, which likely includes the standard SQL. Therefore, it might be familiar with standard text-to-SQL tasks, which is also consistent with the recent finding (Liu et al., 2023).

5.3.2 Effects on Execution time

Details We further evaluate the impact of our special tokens with respect to query execution efficiency. Specifically, we compare the execution time of the queries from EHR-SeqSQL and that of their standard SQL version. The original database consists of the medical records of 1,000 patients and has a size of 95MB. We additionally constructed two larger databases following Lee et al. (2022), one with the medical records of 10,000 patients and another with all 46,520 patients that MIMIC provides. The size of these databases is 921MB and 5.06GB respectively. The results are reported on the queries from the test set in a random split, which covers all types of SQL queries in EHR-SeqSQL[1].

Result The result is shown in Table 5. We observed an 18% decrease in average execution time, in the original database. On a single query basis, the execution time is reduced at most to 99.89%, where the original SQL query has five nested queries inside. The effectiveness of the special token tends to increase with the size of the database. In the largest database, the special token yielded an average time reduction of 81.0%. Given that real-world databases typically contain huge amounts of data, we anticipate that the special token will yield a significant practical impact in multi-turn text-to-SQL environments.

6 Conclusion

We present EHR-SeqSQL, the first sequential text-to-SQL dataset designed for Electronic Health Record databases, aimed at improving interactivity, compositionality, and efficiency in text-to-SQL parsing. In terms of compositionality, our experiments show that the decomposition of questions significantly enhances the model's ability to generalize to unseen interaction goals. Furthermore, it exhibited the potential for generalizing longer interactions that were not encountered during training. Additionally, we propose novel SQL-like tokens that enhance execution efficiency by reusing query execution results from the previous turn, as well as improve the performance of sequential text-to-SQL tasks by reducing the decoding burden on the model. As a step forward to interactive settings in text-to-SQL, we believe our dataset will serve as a valuable testbed for assessing contextual and interactive text-to-SQL tasks in EHRs and bridge the gap between industry needs and academic research.

[1]We excluded the execution time that is nearly zero.

Limitations

Firstly, EHR-SeqSQL is generated through SQL query decomposition from EHRSQL, which may not fully represent real-world use cases. For example, real-world questions are often ambiguous and may contain unanswerable questions given the database. Although handling ambiguous and unanswerable questions is not within the scope of this work, it is an important topic that we would like to explore in the future. Secondly, our dataset assumes users possess a certain level of understanding of the database schema. For example, when asking about a patient's records, users must first identify the specific hospitalization ID for that patient. This assumption could limit the applicability of our dataset in scenarios where users lack this level of comprehension or knowledge. We note, however, that in domain-specific settings, users are typically aware of the database's structure to certain extent.

Ethics Statement

Our main concerns are patient privacy and ensuring that the proposed dataset as well as the construction process are both compliant with the privacy regulations. The MIMIC-III database, which requires credentialed access via PhysioNet, is a database that contains de-identified medical records of intensive care patients. Lee et al. (2022) further de-identified MIMIC-III for EHRSQL by shifting time and values across the database. We use the database constructed from EHRSQL, which carefully prevents the re-identification of medical records from the questions. We utilized the ChatGPT api to paraphrase our manually generated question templates. Each question template corresponds to the decomposed queries from EHRSQL and would unlikely contain the sensitive medical information.

References

Jayetri Bardhan, Anthony Colas, Kirk Roberts, and Daisy Zhe Wang. 2022. Drugehrqa: A question answering dataset on structured and unstructured electronic health records for medicine related queries. *arXiv preprint arXiv:2205.01290*.

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. *arXiv preprint arXiv:1806.09029*.

Yujian Gan, Xinyun Chen, Qiuping Huang, and Matthew Purver. 2022. Measuring and improving compositional generalization in text-to-sql via component alignment. *arXiv preprint arXiv:2205.02054*.

Biyang Guo, Xin Zhang, Ziyuan Wang, Minqi Jiang, Jinran Nie, Yuxuan Ding, Jianwei Yue, and Yupeng Wu. 2023. How close is chatgpt to human experts? comparison corpus, evaluation, and detection. *arXiv preprint arXiv:2301.07597*.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*.

Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021. Text-to-sql in the wild: a naturally-occurring dataset based on stack exchange data. *arXiv preprint arXiv:2106.05006*.

Charles T Hemphill, John J Godfrey, and George R Doddington. 1990. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. 2022. Ehrsql: A practical text-to-sql benchmark for electronic health records. *Advances in Neural Information Processing Systems*, 35:15589–15601.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111*.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*.

Tom J Pollard, Alistair EW Johnson, Jesse D Raffa, Leo A Celi, Roger G Mark, and Omar Badawi. 2018. The eicu collaborative research database, a freely available multi-center database for critical care research. *Scientific data*, 5(1):1–13.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Tianze Shi, Jonathan Herzig, Emily Pitler, Fei Sha, and Kristina Toutanova. 2022. Evaluating the impact of model scale for compositional generalization in semantic parsing. *arXiv preprint arXiv:2205.12253*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*.

Ping Wang, Tian Shi, and Chandan K Reddy. 2020. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361.

Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. 2021. Data augmentation with hierarchical sql-to-question generation for cross-domain text-to-sql parsing. *arXiv preprint arXiv:2103.02227*.

Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. 2019a. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. *arXiv preprint arXiv:1909.05378*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, et al. 2019b. Sparc: Cross-domain semantic parsing in context. *arXiv preprint arXiv:1906.02285*.

Rong Zhou. 2022. Research on key performance index prediction of distributed database based on machine learning algorithm. In *International Conference on Cognitive based Information Processing and Applications*, pages 563–567. Springer.

Xuanhe Zhou, Guoliang Li, Chengliang Chai, and Jianhua Feng. 2021. A learned query rewrite system using monte carlo tree search. *Proceedings of the VLDB Endowment*, 15(1):46–58.

A Data Construction Details

Category	Example	# of questions
Independent	Q1. What is the specific item id of the hemoglobin lab test?	14,005 (44.22%)
Dependent		
Referential	Q1. Display the hospitalization ids of patient 76173. Q2. How many times each drug was prescribed within A1 since 2101?	11,560 (36.50%)
Filtering	Q1. What are the calcium, total lab test values tested during A1? Q2. Retrieve the last tested case in A2.	5,947 (18.78%)
Modifying	Q1. During A1, what was the last measured value of A2? Q2. What about the first measured case?	507 (1.16%)

Table 6: Category of the questions within EHR-SeqSQL.

A.1 Details in Stage 2 in Section 3.1

In stage 2, we decompose the subqueries based on the SQL clauses. Decomposed clauses are parsed according to the logical order of execution of an SQL statement - WHERE, ORDER BY, HAVING, and SELECT. However, there are certain cases that are excluded from this decomposition. First, if a WHERE clause contains any aliases of table or column names without specifying the original name, it is not decomposed for clarity. For example, see `SELECT DISTINCT T1.C1 FROM ( PREV_QUERY5 ) AS T1 WHERE` **`T1.valuenum = 73.0`**. If decomposed, the meaning of `T1` would be ambiguous since we do not know the meaning of T1, which is not a column name in MIMIC-III. Thus we ensured the conditions or clauses with table aliases are always used with the subquery where the alias is defined. Second, we do not decompose the clauses for optional data cleansing SQL clauses. For example, see `SELECT microbiology events.org_name FROM microbiologyevents WHERE microbiologyevents.spec_type_desc = 'foot culture' AND` **`microbiologyevents.org_name IS NOT NULL`**. In this SQL query, the bolded subquery is intended to ensure all the selected rows have contents, excluding any NULL values. It is an optional condition and a SQL writing style choice which is not included in the original question, so we choose not to decompose such clauses. Also, the ten shortest question templates from EHRSQL are maintained without further splitting or modifications, which are not likely to be asked through multiple sentences. These include questions asking about the drug intake method, the cost of a lab test, or the number of current patients. We do not decompose `GROUP BY` but combine it either with `SELECT` or `HAVING`, since they are not explicitly expressed in the natural language questions (Guo et al., 2019; Wu et al., 2021).

A.2 Details in Stage 3 in Section 3.1

The BPE algorithm was applied to merge the subqueries from both Stage 1 and Stage 2. Specifically, we first derived SQL templates by masking condition values in the SQL queries obtained in Stage 2. Each template was treated as a token in the BPE algorithm, and the syntactically mergeable pairs among the most frequent token bigrams were repeatedly merged. A syntactically mergeable pair refers to a pair of queries that were originally a single SQL query but were separated due to our decomposition strategy. Since BPE algorithm calculates frequency without considering the relationship of token bigrams, we added this constraint to make every resultant SQL query to be executable. Then, we sampled half of each randomly to maximize the complexity of each subquery and the diversity of each interaction. Considering the total number of turns in each stage, we merged bigrams that appear more than 100 times in the subqueries from Stage 1 and bigrams that appear more than 150 times in the subqueries from Stage 2. The final SQL queries for EHR-SeqSQL are acquired throughout three stages.

Split	Random			Compositional		
	Train	Test	Test$_L$	Train	Test	Test$_L$
# of Question Temp	167	166	166	121	46	46
# of Interactions	8,546	649	100	6,375	2,820	100
# of Turns	29,438	2,231	1,417	22,134	9,535	1,564
Avg. # of Turns	3.44	3.44	14.17	3.47	3.38	15.64
Max # of Turns	9	9	25	8	8	27

Table 7: Statistics for two different splits.

A.3 List of NLQ templates

In Section 3.2, we mentioned the process for rule-based SQL-to-NLQ generation. Detailed steps of this process can be found in Table 8. We used specific SQL templates paired with NLQ templates for this process, as presented in Table 9. Each placeholder in the NLQ template is determined by the DB schema and the values present in the actual SQL query. A few examples showcasing the pairing of DB schema and NL expressions are available in Table 10. The placeholders related to *value*, *operation*, and *time* are used in the same way as described by Lee et al. (2022).

A.4 Prompt for paraphrasing

Figure 3 shows the prompt for paraphrasing the questions at an interaction level. For each NLQ template, key expressions (such as condition values, reference indices, etc.) that must be strictly preserved were indicated in order to maintain consistency. During the data construction process, we utilized the ChatGPT API to paraphrase template questions. Prior to paraphrasing, all specific values within each template are replaced with representative, generic values for their respective slots. Once paraphrased, these generic values are then realized back to their original form. This process naturally prevents any sensitive information from being sent to the ChatGPT server.

B Details on the Compositional Split

The concept of compositions and components is based on the SQL query since it is the common factor across an EHRSQL instance and its corresponding EHR-SeqSQL instance. Table 11 provides three examples of composition and components. There are EHRSQL questions which is an interaction goal of the corresponding interaction in EHR-SeqSQL, and their templates where the same anonymizing logic in deriving composition is applied.

We use a greedy algorithm to automatically split our data into training and test sets. Starting with all compositions assigned to the training set, we iteratively allocate a composition that has the maximum number of unique components to the test set while constraining all the components in the composition to exist in the training set, until no composition can be further assigned to the test set. You can find that the components in the last row are all present in the components of the first two rows. Thus, according to

Prompt for paraphrasing

---NLQ of text-to-SQL sequence.
<Our NLQ template> e.g.,Retrieve only the last taken case today from result3.

Our objective is to substitute each natural language query (NLQ) turn in a contextual text-to-SQL sequence with a paraphrased sentence that maintains the same meaning using alternate vocabulary or word order.
It is essential to ensure that the semantic coherence of each turn with SQL is preserved.
Also, each sentence should not be too long. Please write concisely and clearly.
Please make 20 different NLQs for the question.
You never ever change these words: **<Key expression>** e.g., ['last', 'today', 'result3']

Figure 3: Prompt for paraphrasing.

interaction goal	What were the top four frequent drugs that patients were prescribed within the same month after having been prescribed with nateglinide last year?			
idx	***SQL Query***	***SQL Template***	***NLQ Template***	***Generated NLQ***
1	SELECT admissions.subject_id, prescriptions.startdate, FROM prescriptions JOIN admissions ON prescriptions.hadm_id = admissions.hadm_id WHERE prescriptions.drug = 'nateglinide' [time_filter_global1]	SELECT table.column, table.column FROM table JOIN table ON table.column = table.column WHERE table.column = [val_placeholder] [time_filter_global1]	List all [SELECT.col:admission.subject_id] and their [SELECT.col:prescriptions.startdate] associated with [val_placeholder:nateglinide] [time_filter_global1:last year].	List all patient ids and their prescription time associated with nateglinide last year.
2	SELECT admissions.subject_id, prescriptions.drug, prescriptions.startdate, FROM prescriptions JOIN admissions ON prescriptions.hadm_id = admissions.hadm_id WHERE [time_filter_global1]	SELECT table.column, table.column, table.column FROM table JOIN table ON table.column = table.column WHERE [time_filter_global1]	List all [SELECT.col:admission.subject_id] and their [SELECT.col:prescriptions.drug] and [SELECT.col:prescriptions.startdate] [time_filter_global1:last year].	List all patient ids and their drugs and prescription time last year.
3	SELECT T2.drug, DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS C1 FROM ([PREV_QUERY1]) AS T1 JOIN ([PREV_QUERY2]) AS T2 ON T1.subject_id = T2.subject_id WHERE T1.startdate <T2.startdate [time_filter_within] GROUP BY T2.drug	SELECT table.column, DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS column FROM ([PREV1]) AS table JOIN ([PREV2]) AS table ON table.column = table.column WHERE table.column <table.column [time_filter_within] GROUP BY table.column	List the frequency rankings of [PREV:2] that patients received [time_filter_within:within the same month] after the [PREV:1].	List the frequency rankings of A2 that patients received within the same month after A1.
4	SELECT T3.drug FROM ([PREV_QUERY3]) AS T3 WHERE T3.C1 <= [n_rank]	SELECT table.column FROM ([PREV3]) AS table WHERE table.column <= [n_rank]	List the top [n_rank:four] [SELECT.col:prescriptions.drug] in [PREV:3].	List the top four drugs in A3.

Table 8: SQL-to-NLQ Generation Process.

SQL template	NLQ template
SELECT ([PREV0]) - ([PREV1])	What is the difference between the [PREV0] and [PREV1]?
SELECT ([PREV0]) [comparison] ([PREV1])	Is [PREV0] [comparison] than [PREV1]?
SELECT [agg_function](table.column) FROM ([PREV]) AS table	What is the [SELECT.col] of [PREV]?
SELECT COUNT(DISTINCT table.column) FROM ([PREV]) AS table	Count the number of patients in [PREV].
SELECT COUNT(DISTINCT table.column) FROM table #cond_parsed	Count the number of [PREV-1].
SELECT COUNT(DISTINCT table.column) FROM table WHERE [time_filter_global1]	Count the number of [PREV-1].
SELECT COUNT(*) FROM table WHERE table.column = ([PREV])	Count the number of [SELECT.col] associated with [PREV].
SELECT COUNT(*) FROM table WHERE table.column = [val_placeholder]	Count the number of [val_placeholder].
SELECT COUNT(*) FROM table WHERE table.column IN ([PREV])	Count the number of [SELECT.col] associated with [PREV].
SELECT COUNT(*)>0 FROM table WHERE table.column = [val_placeholder]	Has [val_placeholder] been admitted to the hospital?
SELECT COUNT(*)>0 FROM table WHERE table.column IN ([PREV])	Are there any [SELECT.col] in [PREV]?
SELECT SUM(table.column) FROM table WHERE table.column IN ([PREV])	What is the [SELECT.col] associated with [PREV]?
SELECT SUM(table.column) FROM table WHERE table.column IN ([PREV])	What is the total amount of [PREV-1]?
SELECT table.column FROM ([PREV]) AS table WHERE table.column [n_times]	Which [SELECT.col] is [n_times] in [PREV]?
SELECT table.column FROM ([PREV]) AS table WHERE table.column <= [n_rank]	List top [n_rank] [SELECT.col] in [PREV].
SELECT table.column FROM table	List all [SELECT.col] from [FROM.table].
SELECT table.column FROM table WHERE [age_group]	List all [SELECT.col] associated with patients aged [age_group].
SELECT table.column FROM table WHERE table.column = ([PREV])	What is the [SELECT.col] of [PREV]?
SELECT table.column FROM table WHERE table.column = [val_placeholder]	List all [SELECT.col] of [val_placeholder].
SELECT table.column FROM table WHERE table.column IN ([PREV])	List all [SELECT.col] associated with [PREV].
SELECT table.column, table.column FROM table	List all [SELECT.col.0] and [SELECT.col.1].
[PREV] [time_filter_exact1]	What was the [time_filter_exact2] measured case from [PREV-1]?
[PREV] [time_filter_global1_dec1]	Retrieve only the cases [time.verb] [time_filter_global1_dec1] from [PREV].
[PREV] [time_filter_global1_dec2]	Retrieve only the cases [time.verb] [time_filter_global1_dec2] from [PREV].
[PREV] [time_filter_global1]	Retrieve only the cases [time.verb] [time_filter_global1] from [PREV].
[PREV] AND [age_group]	Retrieve only the cases associated with patients aged [age_group] from [PREV_QEURY].
[PREV] AND table.column = ([PREV])	Retrieve only the cases associated with [PREV] from [PREV].
[PREV] AND table.column IN ([PREV])	Retrieve only the cases associated with [PREV] from [PREV].
[PREV] AND table.column IS NULL	What is the current one in [PREV]?
[PREV] WHERE [age_group]	Retrieve only the cases associated with patients aged [age_group] from [PREV].
[PREV] WHERE [time_filter_global1]	Retrieve only the cases [time.verb] [time_filter_global1] from [PREV].
[PREV] WHERE table.column = ([PREV])	Retrieve only the cases associated with [PREV] from [PREV].
[PREV] WHERE table.column IN ([PREV])	Retrieve only the cases associated with [PREV] from [PREV_QEURY].

Table 9: SQL & NLQ template.

DB schema	NL expression
admissions.admittime	admission time
admissions.dob	date of birth
admissions.dod	date of death
admissions.subject_id	patient
chartevents.charttime	chart time
chartevents.itemid	vital sign item id
chartevents.valuenum	value of vital sign
cost.hadm_id	hospital stay
diagnoses_icd.charttime	time of diagnosis
diagnoses_icd.icd9_code	diagnosis
diagnoses_icd.icd9_code	diagnosis ICD-9 code
inputevents_cv.amount	volume of intake
inputevents_cv.itemid	input event item id
labevents.itemid	lab test
labevents.itemid	lab test item id
labevents.valuenum	value of lab test
microbiologyevents.org_name	organism name
microbiologyevents.spec_type_desc	microbiology test
outputevents.itemid	output event item id
prescriptions.drug	drug
prescriptions.startdate	prescription time
procedures_icd.charttime	time of procedure
procedures_icd.hadm_id	hospital stay
procedures_icd.icd9_code	procedure ICD-9 code
procedures.icd9_code	procedure

Table 10: Examples of DB schema & NL expression pairs.

Interaction Goal	Template	Composition	Set of Components
Is the value of glucose of patient 71192 last measured on the first hospital visit less than the second to last value measured on the first hospital visit?	Is the value of {lab_name} of patient {patient_id} [time_filter_exact2] measured [time_filter_global2] [comparison] than the [time_filter_exact1] value measured [time_filter_global1]?	SELECT (SELECT labevents.valuenum FROM labevents WHERE labevents.hadm_id IN (SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = {patient_id} [time_filter_global1]) AND labevents.itemid IN (SELECT d_labitems.itemid FROM d_labitems WHERE d_labitems.label = {lab_name}) [time_filter_exact1]) < (SELECT labevents.valuenum FROM labevents WHERE labevents.hadm_id in (SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = {patient_id} [time_filter_global2]) AND labevents.itemid IN (SELECT d_labitems.itemid FROM d_labitems WHERE d_labitems.label = {lab_name}) [time_filter_exact2])	1. SELECT admissions.hadm_id FROM admissions 2. [PREV_QUERY] WHERE admissions.subject_id = {patient_id} 3. [PREV_QUERY] AND [time_filter_global1] 4. SELECT d_labitems.itemid FROM d_labitems 5. [PREV_QUERY] WHERE d_labitems.label = {lab_name} 6. SELECT labevents.valuenum FROM labevents 7. [PREV_QUERY] WHERE labevents.hadm_id IN ([PREV_RESULT]) 8. [PREV_QUERY] AND labevents.itemid IN ([PREV_RESULT]) 9. [PREV_QUERY] [time_filter_exact1] 10. [PREV_QUERY] [time_filter_exact2] 11. SELECT ([PREV_RESULT]) [comparison] ([PREV_RESULT])
What was the maximum arterial bp [diastolic] of patient 18866 yesterday?	What was the [agg_function] {vital_name} of patient patient_id [time_filter_global1]?	SELECT [agg_function](chartevents.valuenum) FROM chartevents WHERE chartevents.icustay_id IN (SELECT icustays.icustay_id FROM icustays WHERE icustays.hadm_id IN (SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = patient_id)) AND chartevents.itemid IN (SELECT d_items.itemid FROM d_items WHERE d_items.label = {vital_name} AND d_items.linksto = 'chartevents') [time_filter_global1]	1. SELECT admissions.hadm_id FROM admissions 2. [PREV_QUERY] WHERE admissions.subject_id = {patient_id} 3. SELECT icustays.icustay_id FROM icustays 4. [PREV_QUERY] WHERE icustays.hadm_id IN ([PREV_RESULT]) 5. SELECT d_items.itemid FROM d_items 6. [PREV_QUERY] WHERE d_items.label = {vital_name} AND d_items.linksto = 'chartevents' 7. SELECT chartevents.valuenum FROM chartevents 8. [PREV_QUERY] WHERE chartevents.icustay_id IN ([PREV_RESULT]) 9. [PREV_QUERY] AND chartevents.itemid IN ([PREV_RESULT]) 10. [PREV_QUERY] [time_filter_global1] 11. SELECT [agg_function](chartevents.valuenum) FROM chartevents WHERE chartevents.icustay_id IN ([PREV_RESULT]) AND chartevents.itemid IN ([PREV_RESULT]) [time_filter_global1]
Is the arterial bp [diastolic] of patient 25461 last measured on the last icu visit greater than the second to last value measured on the last icu visit?	Is the {vital_name} of patient {patient_id} [time_filter_exact2] measured [time_filter_global2] [comparison] than the [time_filter_exact1] value measured [time_filter_global1]?	SELECT (SELECT chartevents.valuenum from chartevents WHERE chartevents.icustay_id IN (SELECT icustays.icustay_id from icustays WHERE icustays.hadm_id IN (SELECT admissions.hadm_id from admissions WHERE admissions.subject_id = {patient_id}) [time_filter_global1]) AND chartevents.itemid IN (SELECT d_items.itemid from d_items WHERE d_items.label = {vital_name} AND d_items.linksto = 'chartevents') [time_filter_exact1]) > (SELECT chartevents.valuenum from chartevents WHERE chartevents.icustay_id IN (SELECT icustays.icustay_id from icustays WHERE icustays.hadm_id IN (SELECT admissions.hadm_id from admissions WHERE admissions.subject_id = {patient_id}) [time_filter_global2]) AND chartevents.itemid IN (SELECT d_items.itemid from d_items WHERE d_items.label = {vital_name} AND d_items.linksto = 'chartevents') [time_filter_exact2])	1. SELECT admissions.hadm_id FROM admissions 2. [PREV_QUERY] WHERE admissions.subject_id = {patient_id} 3. SELECT icustays.icustay_id FROM icustays 4. [PREV_QUERY] WHERE icustays.hadm_id IN ([PREV_RESULT]) 5. [PREV_QUERY] AND [time_filter_global1] 6. SELECT d_items.itemid FROM d_items 7. [PREV_QUERY] WHERE d_items.label = {vital_name} AND d_items.linksto = 'chartevents' 8. SELECT chartevents.valuenum FROM chartevents 9. [PREV_QUERY] WHERE chartevents.icustay_id IN ([PREV_RESULT]) 10. [PREV_QUERY] AND chartevents.itemid IN ([PREV_RESULT]) 11. [PREV_QUERY] [time_filter_exact1] 12. [PREV_QUERY] [time_filter_exact2] 13. SELECT ([PREV_RESULT]) [comparison] ([PREV_RESULT])

Table 11: Examples of compositions and components.

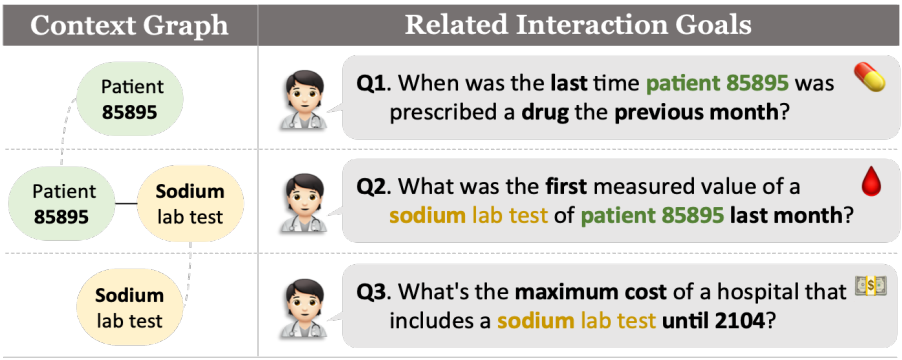

Figure 4: Related interaction goals and their context graph.

our split algorithm, if both the first two compositions are in the training set, the last composition can be assigned to the test set.

C Longer Interaction

C.1 Generation Process

As given in Figure 4, we define the concept of related interaction goal by using the context graph. Each interaction goal has its own context graph, whose nodes are defined by the specific condition values (e.g. conditions for patient, drug, or lab tests) in the original EHRSQL. Two questions are deemed related if they have overlapping condition values. Q1 and Q2 are related because they share the same patient 85895. However, Q1 and Q3 are not directly related without Q2 as a bridge. While we used three independent questions from the EHRSQL to demonstrate this concept in Figure 4, a longer and context-dependent interaction used in Section 5.2 is depicted in Figure 5. In this example for longer interaction, five EHRSQL questions are connected, each paired with an original SQL query. In contrast, EHR-SeqSQL involves context-dependent interaction spanning twelve turns, each paired with a query with our special tokens for SQL.

C.2 IFF Score Calculation

$$IFF = \begin{cases} n+1, & \text{if all turns are correct} \\ k, & \text{otherwise} \end{cases}$$

k denotes the specific turn number in the interaction where the first incorrect response occurs. n represents the total number of turns in the interaction. The final IFF score for a test set is calculated as the average IFF score across all interactions. Hence, if the model achieves a perfect score for every interaction, the IFF score becomes one plus the average number of turns of all interactions.

Note that the perfect IFF score for QQ and QS setting is different in Table 3. This is because we removed the test sample that exceeds the maximum token length of T5 when the interaction history is concatenated with the current question for a fair comparison.

D Fine-tuning Baseline

D.1 Configuration

We fine-tuned the T5-Base and T5-3B using the Adam optimizer, with a global batch size of 32. The learning rate were set to 1e-4 for T5-Base and 5e-5 for T5-3B, respectively. For the other hypterparameter configurations, we followed the settings used in EHRSQL. All experiments were carried out on either a single A100 80G GPU or a A6000 48G GPU. The training process typically took around 10 hours for T5-Base and around 24 hours for T5-3B.

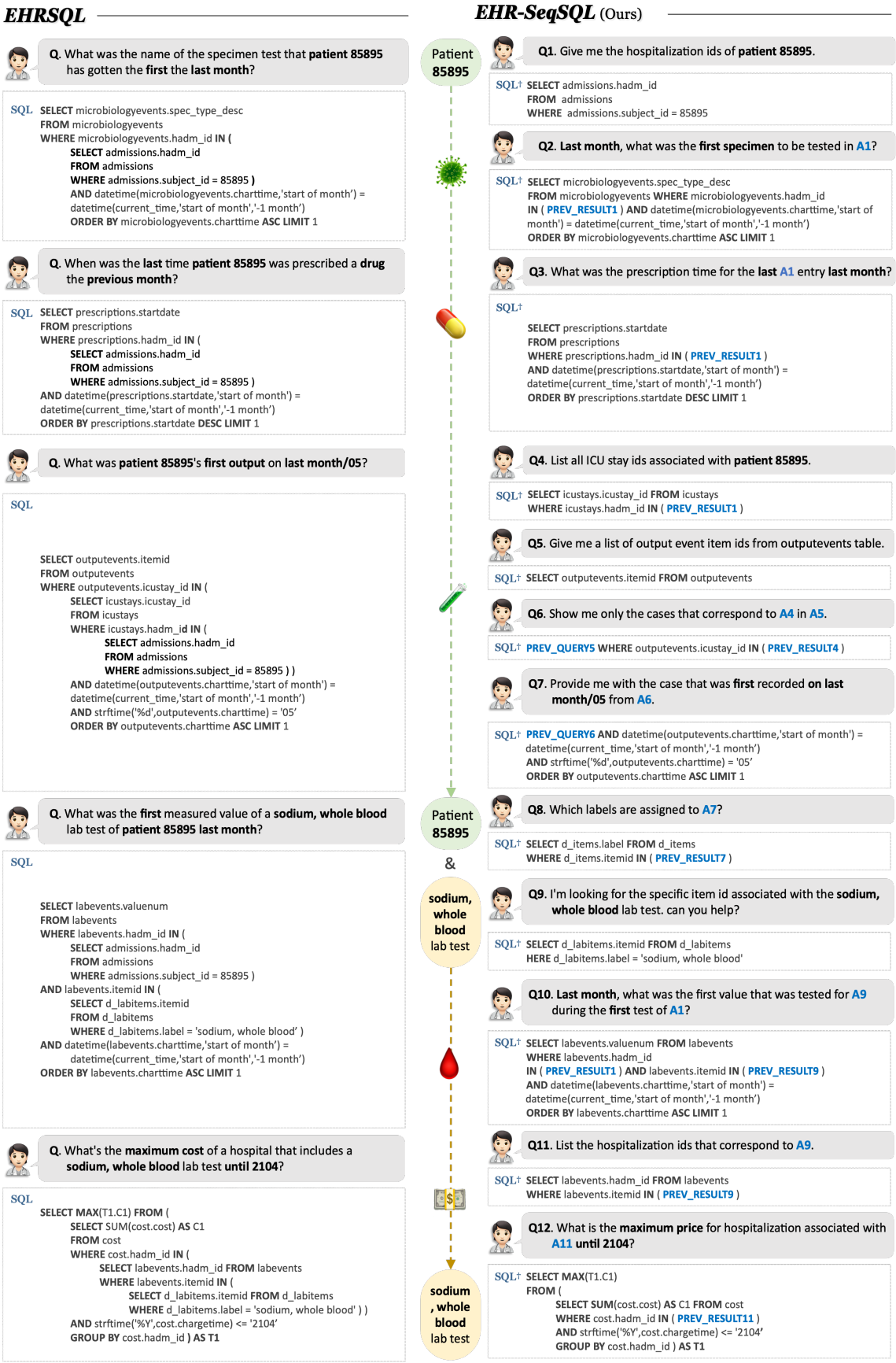

Figure 5: Example of Longer Interaction.

T5-base																
Random									Compositioinal							
#	IM				QM				#	IM				QM			
	TRUE	cnt	%		TRUE	cnt	%			TRUE	cnt	%		TRUE	cnt	%	
1					97	100	97.00%		1					96	100	96.00%	
2					100	100	100.00%		2					83	100	83.00%	
3					96	100	96.00%		3					83	100	83.00%	
4					63	100	63.00%		4					61	100	61.00%	
5					58	100	58.00%		5					60	100	60.00%	
6					48	100	48.00%		6					54	100	54.00%	
7					53	100	53.00%		7					40	100	40.00%	
8					48	100	48.00%		8					42	100	42.00%	
9					62	100	62.00%		9					43	100	43.00%	
10					45	100	45.00%		10					42	100	42.00%	
11					40	100	40.00%		11					39	100	39.00%	
12	0	24	0.00%		24	100	24.00%		12	1	15	6.67%		34	100	34.00%	
13	0	24	0.00%		18	76	23.68%		13	0	17	0.00%		31	85	36.47%	
14	0	16	0.00%		12	52	23.08%		14	0	19	0.00%		18	68	26.47%	
15	0	16	0.00%		1	36	2.78%		15	0	8	0.00%		19	49	38.78%	
16	0	6	0.00%		0	20	0.00%		16	0	8	0.00%		16	41	39.02%	
17	0	3	0.00%		1	14	7.14%		17	0	6	0.00%		7	33	21.21%	
18	0	9	0.00%		1	11	9.09%		18	0	8	0.00%		8	27	29.63%	
19	0	1	0.00%		0	2	0.00%		19	0	7	0.00%		7	19	36.84%	
20					0	1	0.00%		20	0	1	0.00%		2	12	16.67%	
21					0	1	0.00%		21	0	4	0.00%		3	11	27.27%	
22					0	1	0.00%		22	0	2	0.00%		2	7	28.57%	
23					0	1	0.00%		23	0	1	0.00%		1	5	20.00%	
24					0	1	0.00%		24	0	3	0.00%		0	4	0.00%	
25	0	1	0.00%		0	1	0.00%		25					0	1	0.00%	
									26					0	1	0.00%	
									27	0	1	0.00%		0	1	0.00%	
	0	**100**	**0.00%**		**767**	**1417**	**54.13%**			**1**	**100**	**1.00%**		**791**	**1564**	**50.58%**	

Figure 6: T5-Base performance on Longer Interactions

ChatGPT																
Random									Compositioinal							
#	IM				QM				#	IM				QM			
	TRUE	cnt	%		TRUE	cnt	%			TRUE	cnt	%		TRUE	cnt	%	
1					100	100	100.00%		1					99	100	99.00%	
2					100	100	100.00%		2					86	100	86.00%	
3					96	100	96.00%		3					67	100	67.00%	
4					94	100	94.00%		4					73	100	73.00%	
5					86	100	86.00%		5					74	100	74.00%	
6					89	100	89.00%		6					72	100	72.00%	
7					85	100	85.00%		7					67	100	67.00%	
8					86	100	86.00%		8					69	100	69.00%	
9					88	100	88.00%		9					72	100	72.00%	
10					96	100	96.00%		10					72	100	72.00%	
11					98	100	98.00%		11					71	100	71.00%	
12	14	24	58.33%		93	100	93.00%		12	11	15	73.33%		75	100	75.00%	
13	16	24	66.67%		72	76	94.74%		13	10	17	58.82%		62	85	72.94%	
14	8	16	50.00%		51	52	98.08%		14	9	19	47.37%		41	68	60.29%	
15	9	16	56.25%		35	36	97.22%		15	3	8	37.50%		33	49	67.35%	
16	4	6	66.67%		20	20	100.00%		16	3	8	37.50%		30	41	73.17%	
17	2	3	66.67%		14	14	100.00%		17	5	6	83.33%		25	33	75.76%	
18	3	9	33.33%		10	11	90.91%		18	5	8	62.50%		20	27	74.07%	
19	0	1	0.00%		2	2	100.00%		19	5	7	71.43%		14	19	73.68%	
20					0	1	0.00%		20	0	1	0.00%		8	12	66.67%	
21					0	1	0.00%		21	2	4	50.00%		7	11	63.64%	
22					1	1	100.00%		22	1	2	50.00%		6	7	85.71%	
23					1	1	100.00%		23	1	1	100.00%		4	5	80.00%	
24					0	1	0.00%		24	2	3	66.67%		3	4	75.00%	
25	0	1	0.00%		0	1	0.00%		25					0	1	0.00%	
									26					1	1	100.00%	
									27	1	1	100.00%		1	1	100.00%	
	56	**100**	**56.00%**		**1317**	**1417**	**92.94%**			**58**	**100**	**58.00%**		**1152**	**1564**	**73.66%**	

Figure 7: ChatGPT performance on Longer Interactions

EHRSQL 20-shot Prompt

20-shot Examples

------EXAMPLE1
NLQ:what are the four drugs most frequently prescribed during the same hospital visit to the patients aged 60 or above after being diagnosed with neutropenia nos in 2100?
SQL:select t3.drug from (select t2.drug, dense_rank() over (order by count(*) desc) as c1 from (select admissions.subject_id, diagnoses_icd.charttime, admissions.hadm_id from diagnoses_icd join admissions on diagnoses_icd.hadm_id = admissions.hadm_id where diagnoses_icd.icd9_code = (select d_icd_diagnoses.icd9_code from d_icd_diagnoses where d_icd_diagnoses.short_title = 'neutropenia nos') and strftime('%y',diagnoses_icd.charttime) = '2100') as t1 join (select admissions.subject_id, prescriptions.drug, prescriptions.startdate, admissions.hadm_id from ...
------ EXAMPLE2
<19 examples here>

Post-processing Detail

Please note that:
1. Questions asking whether a specific number falls within a normal range can be formulated as follows and will be changed through the post_processing process.
NLQ: Had the value of result5 measured during result4 been normal?
SQL: SELECT COUNT(*)>0 FROM chartevents WHERE chartevents.icustay_id IN (...) AND chartevents.valuenum BETWEEN sao2_lower AND sao2_upper
2. Similarly, for questions that require the current time, we will use 'current_time' as a placeholder and adjust it as necessary. For reference, the current time is assumed to be "2105-12-31 23:59:00". Therefore, if there is the expression "this month" means 2105-12.

Test Question

-- TEST_QUESTION
NLQ:this year, what are the top three most frequent drugs prescribed during the same hospital visit to the patients aged 60 or above after being diagnosed with exam-clincal trial?
SQL:

EHR-SeqSQL 20-shot Prompt

20-shot Examples

------EXAMPLE1
NLQ1:Retrieve the patient ids, hospitalization ids, and diagnosis times for glaucoma nos cases diagnosed last year.
SQL1:SELECT admissions.subject_id, diagnoses_icd.charttime, admissions.hadm_id FROM diagnoses_icd JOIN admissions ON diagnoses_icd.hadm_id = admissions.hadm_id WHERE diagnoses_icd.icd9_code = (SELECT d_icd_diagnoses.icd9_code FROM d_icd_diagnoses WHERE d_icd_diagnoses.short_title = 'glaucoma nos') AND datetime(diagnoses_icd.charttime,'start of year') = datetime(current_time,'start of year','-1 year')
NLQ2:Can you provide the patient ids, drugs, prescription time, and hospitalization ids for patients aged 50s last year?
SQL2:...
NLQ3:...
------ EXAMPLE2
<19 examples here>

Post-processing Detail

Please note that:
1. Questions asking whether a specific number falls within a normal range can be formulated as follows and will be changed through the post_processing process.
NLQ: Had the value of result5 measured during result4 been normal?
SQL: SELECT COUNT(*)>0 FROM chartevents WHERE chartevents.icustay_id IN (...) AND chartevents.valuenum BETWEEN sao2_lower AND sao2_upper
2. Similarly, for questions that require the current time, we will use 'current_time' as a placeholder and adjust it as necessary. For reference, the current time is assumed to be "2105-12-31 23:59:00". Therefore, if there is the expression "this month" means 2105-12.

SQL-like Rep. Description

PREV_QUERY and PREV_RESULT tokens allow for referencing and reusing the SQL code and results of previous queries in subsequent ones.
The PREV_QUERY token is used to represent the SQL code of the previous query, essentially allowing the new query to build upon it or modify it. SQL queries can also start with the PREV_QUERY token, which enables the duplication and utilization of the previous query in the new one.
The PREV_RESULT token, on the other hand, is used to represent the actual result set from a previous query, rather than the query itself. This is useful when we want to use the results of a previous query directly within a new query.

Test Question

-- TEST_QUESTION
NLQ1:Retrieve the patient ids, hospitalization ids, and diagnosis times for exam-clincal trial cases diagnosed this year.
SQL1:

Figure 8: Prompt configuration for chatGPT.

E In-context Learning Baseline

E.1 Prompt Configuration

Figure 8 demonstrates how we configured the prompt of ChatGPT for both the EHRSQL and EHR-SeqSQL. Unlike EHRSQL where the target representation is standard SQL, EHR-SeqSQL include the special tokens, which are first introduced in this work. Thus, we include a simple description for `PREV_QUERY` and `PREV_RESULT` tokens in the prompt. In the *QS* setting during the inference process, for the current turn, the interaction history includes both previous questions and queries generated by ChatGPT.

E.2 Few-Shot Learning Approach for EHR-SeqSQL

We developed a new method for few-shot learning for the multi-turn, context-dependent setting of EHR-SeqSQL. This method is detailed in Figure **??**. To begin, we created two corpora: one for interaction-level training data and another for turn-level training data. For our few-shot learning approach, we retrieved examples, with half based on the interaction history and the other half based on the current question. In our experiments, we used a total of 20 examples for few-shot learning.

F Use Cases of the Special Tokens

Table 12 illustrates an example of the interactions in EHR-SeqSQL where the special token significantly reduce the length of target representation as well as the execution time. Due to readability and spatial issues, the standard SQL version of each turn has been omitted from the table. You can see the standard SQL would be quite lengthy by looking at the SQL† that the used special token refers to.

Index	Question	Target Representation (SQL†)	Time (SQL†)	Standard SQL	Time (SQL)
1	Which icu stay ids are associated with patient 30826 on the current hospital visit?	SELECT icustays.icustay_id FROM icustays WHERE icustays.hadm_id IN (SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = 30826 AND admissions.dischtime IS NULL)	0.334	SELECT icustays.icustay_id FROM icustays WHERE icustays.hadm_id IN (SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = 30826 AND admissions.dischtime IS NULL)	0.334
2	Could you tell me the item id for weight, please?	SELECT d_items.itemid FROM d_items WHERE d_items.label = 'admit wt' AND d_items.linksto = 'chartevents'	0.334	SELECT d_items.itemid FROM d_items WHERE d_items.label = 'admit wt' AND d_items.linksto = 'chartevents'	0.333
3	During result1, what was the last value of result2 that was measured?	SELECT chartevents.valuenum FROM chartevents WHERE chartevents.icustay_id IN (PREV_RESULT1) AND chartevents.itemid IN (PREV_RESULT2) ORDER BY chartevents.charttime DESC LIMIT 1	20.352	SELECT chartevents.valuenum FROM chartevents WHERE chartevents.icustay_id IN (SELECT icustays.icustay_id FROM icustays WHERE icustays.hadm_id IN (SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = 30826 AND admissions.dischtime IS NULL)) AND chartevents.itemid IN (SELECT d_items.itemid FROM d_items WHERE d_items.label = 'admit wt' AND d_items.linksto = 'chartevents') ORDER BY chartevents.charttime DESC LIMIT 1	31.362
4	What about the first measured case?	SELECT chartevents.valuenum FROM chartevents WHERE chartevents.icustay_id IN (PREV_RESULT1) AND chartevents.itemid IN (PREV_RESULT2) ORDER BY chartevents.charttime ASC LIMIT 1	19.684	SELECT chartevents.valuenum FROM chartevents WHERE chartevents.icustay_id IN (SELECT icustays.icustay_id FROM icustays WHERE icustays.hadm_id IN (SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = 30826 AND admissions.dischtime IS NULL)) AND chartevents.itemid IN (SELECT d_items.itemid FROM d_items WHERE d_items.label = 'admit wt' AND d_items.linksto = 'chartevents') ORDER BY chartevents.charttime ASC LIMIT 1	30.695
5	What is the variation between result3 and result4?	SELECT (PREV_RESULT3) - (PREV_RESULT4)	0.000	SELECT (SELECT chartevents.valuenum FROM chartevents WHERE chartevents.icustay_id IN (SELECT icustays.icustay_id FROM icustays WHERE icustays.hadm_id IN (SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = 30826 AND admissions.dischtime IS NULL)) AND chartevents.itemid IN (SELECT d_items.itemid FROM d_items WHERE d_items.label = 'admit wt' AND d_items.linksto = 'chartevents') ORDER BY chartevents.charttime DESC LIMIT 1) - (SELECT chartevents.valuenum FROM chartevents WHERE chartevents.icustay_id IN (SELECT icustays.icustay_id FROM icustays WHERE icustays.hadm_id IN (SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = 30826 AND admissions.dischtime IS NULL)) AND chartevents.itemid IN (SELECT d_items.itemid FROM d_items WHERE d_items.label = 'admit wt' AND d_items.linksto = 'chartevents') ORDER BY chartevents.charttime ASC LIMIT 1)	60.722

Table 12: Example of an interaction in EHR-SeqSQL where the target representations (SQL†) contain special tokens. We also report average execution times in milliseconds(10^{-3}), where the queries are executed three times. Their standard SQL versions are also reported for comparison.