Knowledge Graph-based Question Answering with Electronic Health Records

Junwoo Park   JUNWOO.PARK@KAIST.AC.KR
Youngwoo Cho   CYW314@KAIST.AC.KR
KAIST / Daejeon, South Korea

Haneol Lee   STUDYMODE@YONSEI.AC.KR
Yonsei University / Wonju, South Korea

Jaegul Choo   JCHOO@KAIST.AC.KR
Edward Choi   EDWARDCHOI@KAIST.AC.KR
KAIST / Daejeon, South Korea

Abstract

Question Answering (QA) is a widely-used framework for developing and evaluating an intelligent machine. In this light, QA on Electronic Health Records (EHR), namely EHR QA, can work as a crucial milestone towards developing an intelligent agent in healthcare. EHR data are typically stored in a relational database, which can also be converted to a directed acyclic graph, allowing two approaches for EHR QA: Table-based QA and Knowledge Graph-based QA. We hypothesize that the graph-based approach is more suitable for EHR QA as graphs can represent relations between entities and values more naturally compared to tables, which essentially require JOIN operations. In this paper, we propose a graph-based EHR QA where natural language queries are converted to SPARQL instead of SQL. To validate our hypothesis, we create four EHR QA datasets (graph-based VS table-based, and simplified database schema VS original database schema), based on a table-based dataset MIMICSQL. We test both a simple Seq2Seq model and a state-of-the-art EHR QA model on all datasets where the graph-based datasets facilitated up to 34% higher accuracy than the table-based dataset without any modification to the model architectures. Finally, all datasets are open-sourced[1] to encourage further EHR QA research in both directions.

1. Introduction

Electronic health records (EHR) consist of heterogeneous data (*e.g.,* demographics, diagnosis, medications, and labs) stored typically in a complex relational database (RDB). An agent that can understand EHR and perform complex reasoning is not only an advancement in AI research, but also has great potential in practical applications such as clinical decision support, hospital administration, and medical chatbots. For example, to obtain answers for questions such as *"When was the last time this patient was prescribed with an antihypertensive?"* based on the information stored in the RDB, both medical experts and ordinary users typically use a graphical user interface that provides limited query assistance, or have to manually write queries such as *"select count (prescriptions.timestep) from patients inner join admissions on patients.subject_id = admissions.subject_id inner join prescriptions on admissions.hadm_id = prescriptions.hadm_id where prescriptions.drug = antihypertensive"*. It is laborious for both medical experts and ordinary users to write queries because in order

---

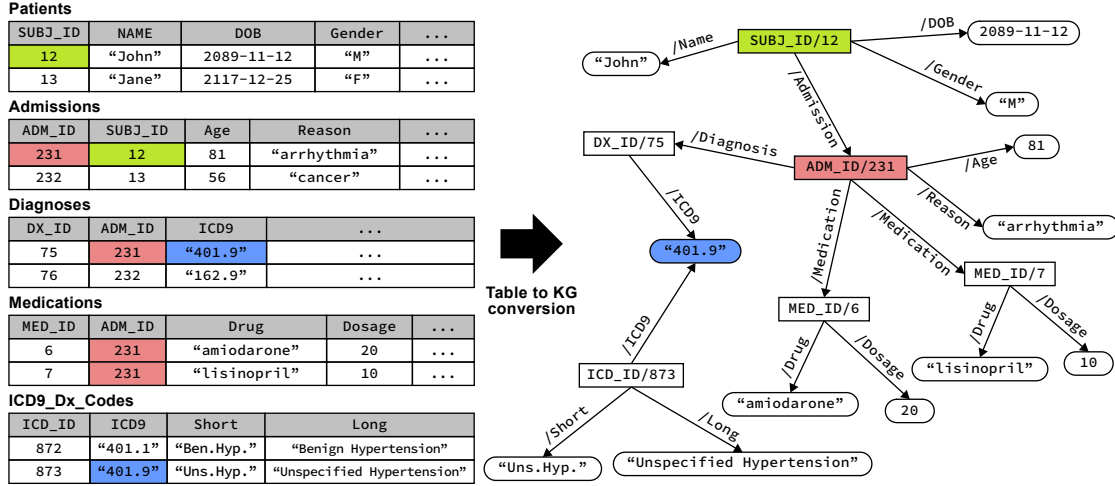1. https://github.com/junwoopark92/mimic-sparql

Figure 1: A subset of tables and their corresponding knowledge graph (best viewed in color). Rectangles and ellipses in the graph indicate entities and literal values, respectively. The color-coded elements are the primary key or the foreign key linking two tables.

to obtain accurate answers, they have to exactly know the hospital RDB schema and the information each table contains, and be proficient in the query language. An intelligent EHR machine that understands the RDB information, the query language, and the natural-language questions (NLQ) can remove significant burden off both experts and ordinary users who seek information from the EHR.

Recently, the Question-Answering (QA) task has played a vital role in developing and evaluating machine intelligence (Bordes et al., 2014; Antol et al., 2015; Seo et al., 2017; Zhong et al., 2017), and we also aim to develop an intelligent EHR agent in a QA framework, namely EHR QA. As for performing QA on RDB data (*i.e.,* tabular data), translating NLQ to SQL (*i.e.,* NLQ2SQL) has become the main approach. This trend can also be seen in EHR QA, as Wang et al. released the first NLQ2SQL dataset for EHR, namely *MIMICSQL*. The RDB, however, stores data typically in a fragmented manner via complex schemas, in order to maximize the efficiency in large-scale data processing and storage. Therefore, SQL has functions such as *JOIN* to aggregate information scattered across multiple tables. This characteristic leads to decreased QA performance of the model trained on NLQ-SQL pairs when the SQL samples are generated from complex schemas.

Although EHR data are typically stored in a relational database, it can be seen as a Directed Acyclic Graph (DAG), or more specifically, a Knowledge Graph (KG) of clinical facts (see Figure 1), thus allowing two approaches for EHR QA: table-based QA and graph-based QA. A knowledge graph is a multi-relational graph composed of entities as nodes and relations among them as typed edges (*e.g.,* John-/medication-amiodarone, amiodarone-/dosage-30ml). SPARQL, which is one of query languages to extract information from KG, supports reasoning over multiple edges of the KG to infer the right answer for the given question. Unlike SQL, SPARQL does not require special functions (*e.g., JOIN*) to collect scattered information, as KG maintains relevant information in a single graph. We actually confirm in the experiment that SPARQL queries is closer to NLQs than SQL queries (Figure 4), and that even if the RDB schema becomes complex, the length of SPARQL

queries does not increase significantly compared to that of SQL queries (Figure 3). Motivated by these findings, we hypothesize that the graph-based approach could be the superior approach for EHR QA, since graphs can represent relations between entities and values more naturally compared to tables, which require *JOIN* operations. Despite such potential of NLQ2SPARQL, to our knowledge, there has not been any prior study exploring the idea of NLQ2SPARQL for EHR QA, and no NLQ-SPARQL pair datasets based on EHR. The goal and contribution of this work can be summarized as follows:

- For an empirical comparison between the two, we construct both a table-based and a graph-based EHR QA datasets based on `MIMICSQL` (Wang et al., 2020), an existing table-based EHR QA dataset. We modify `MIMICSQL` to follow the original schema of MIMIC-III (Johnson et al., 2016), thus building `MIMICSQL*` and its graph-based counterpart `MIMIC-SPARQL*`.

- We test the state-of-the-art EHR QA model called TREQS (Wang et al., 2020), on both `MIMICSQL*` and `MIMIC-SPARQL*` where TREQS gained a 5.1% boost in predicting the correct relations between entities, and a 3.6% boost in predicting the correct answer.

- We carefully analyze the reason for the graph-based method's superiority over the table-based method, presenting histograms in which the overall length of SQLs increases when the RDB schema becomes complex, and an experiment in which the number of correct SQLs decreases when the number of *JOIN*s increases compared to SPARQL.

- To the best of our knowledge, this is the first work to propose and empirically demonstrate the graph-based EHR QA. In order to encourage further EHR QA research in both the table-based and graph-based directions, we open-source both `MIMICSQL*` and `MIMIC-SPARQL*`.

Generalizable Insights about Machine Learning in the Context of Healthcare

This work is closely related to healthcare from three perspectives. First, the development of an agent that can understand EHR and interact with humans in natural language can provide various conveniences to healthcare workers and consumers. Second, through the task of QA, we can quantitatively evaluate how well the machine understands medical data. Lastly, by extending the table QA dataset to the graph QA dataset, we can explore what is more suitable data structure for performing QA on the complex EHR.

2. Background

2.1. Electronic Health Records QA

QA is a discipline in the fields of information retrieval and natural language processing, concerned with building systems that automatically answer questions posed by humans in natural language. In the medical domain especially, EHR QA is a research area with great potential for real-world impact, as it provides an interface where both medical experts and ordinary users can easily query clinical facts and statistics via natural language.

Recent QA research can generally be categorized into unstructured and structured QAs. The former mainly handles free-text, both unimodal (*e.g.,* machine reading comprehension (MRC) datasets (Rajpurkar et al., 2016; Nguyen et al., 2016)) and multi-modal (*e.g.,* Visual Question Answering

(VQA) (Antol et al., 2015; Johnson et al., 2017)). Structured QA can largely be divided into table-based QA (Neelakantan et al., 2015; Zhong et al., 2017) and graph-based QA (Berant et al., 2013; Yih et al., 2015). EHR generally consist of multiple sources of information such as patient demographics, diagnosis history, medication records, lab results, clinical notes, radiology images, etc. Such heterogeneous information is usually stored in a relational database, which can also be seen as a Directed Acyclic Graph (DAG) of clinical fact, or more specifically a Knowledge Graph (see, for example, Figure 1). Therefore, we view EHR QA as a type of structured QA, allowing both a table-based and a graph-based approaches.

Previously, Pampari et al. constructed emrQA, an MRC dataset based on clinical notes.emrQA, however, focuses only on the free-text notes, thus not suitable for training machines to leverage the structural properties of EHR. Recently, (Wang et al., 2020) used MIMIC-III (Johnson et al., 2016), a publicly available EHR dataset, to construct MIMICSQL, a table-based (*i.e.,* text-to-SQL) QA dataset. However, while constructing MIMICSQL, they preprocessed the tables of MIMIC-III such that the original schema was heavily modified. Therefore the final dataset (questions and tables) does not reflect the complex, yet interesting structure of MIMIC-III, thus losing real-world applicability and the value as a tool for fostering machine intelligence.

2.2. Question-to-Query Generation

To solve a structured QA, the widely used approach is a Natural Language Question-to-Query (NLQ2Query) generation, which is a sub-task of semantic parsing. The semantic parsing aims at translating a natural language text to a logical form, which is a formal semantic representation of the given natural text. For various applications, it is desirable for the logical forms to be immediately executable by the applications, such as SQL queries in the database and SPARQL queries in the knowledge graph.

Recently, NLQ2SQL generation has attracted significant attention and demonstrated successful results in various applications, including WikiSQL (Zhong et al., 2017; Xu et al., 2017) for Wikipedia, GeoQuery (Finegan-Dollak et al., 2018) for US geograpy and Spider (Yu et al., 2018) for domain generalization. In the literature of NLQ2SQL, the standard approach is the sequence-to-sequence (Seq2Seq) based method. Seq2Seq based methods (Sutskever et al., 2014; Dong and Lapata, 2016; McCann et al., 2018) have an encoder and a decoder, where both components use a sequence model such as the LSTM. More specifically, Seq2Seq based methods, which are trained with NLQ-SQL pairs, first encode input questions into vector representations and then decode the corresponding SQL queries conditioned on the encoded vectors.

Naturally, these successes led to the work (Wang et al., 2020) in the medical field with abundant EHR data. Note that the wide use of medical jargons and abbreviations create significant challenge, making it difficult to translate natural questions to the SQL form. They cause various problems including difficult decoding and the Out-of-Vocabulary (OOV), where the model has to translate words that are unseen in training dataset. To deal with the OOV problem, a pointer generating mechanism (See et al., 2017) is added to the Seq2Seq methods, where the decoder generates placeholder tokens and copies a word from the input question to substitute the placeholder token. TREQS is the state-of-the-art model proposed by Wang et al. for EHR QA, which uses the pointer generating mechanism and two attention mechanisms: the temporal attention on questions, and the dynamic attention on SQL, and attentive-copy techniques. This model is able to handle the unique challenges

Table 1: Basic statistics of `MIMICSQL*` and `MIMIC-SPARQL*`. The values in the parentheses are from the original `MIMICSQL` and its counterpart `MIMIC-SPARQL`.

	`MIMICSQL*`	`MIMIC-SPARQL*`
# of patients		100
# of Question-Query pairs		10,000
Avg. TP question length		18.40
Avg. NL question length		16.45
# of unique words in TP questions	6,696 (6,693)	6,525 (6,525)
# of unique words in NL questions	7,331 (7,329)	7,160 (7,160)
Avg. SQL/SPARQL length	44,68 (21.04)	28.98 (27.28)
Max # of *INNER JOIN*s	5 (2)	-
Max depth of graph	-	5 (3)
# of triples	-	173,096 (257,187)

in EHR QA and is robust to the unseen questions. TREQS demonstrated its effectiveness on the EHR NLQ2SQL dataset (`MIMICSQL`).

Compared to the table-based QA where information is stored in tables, we can think of graph-based QA where information is stored in knowledge graphs (KG). Recently, QA over KG (KGQA) is actively studied and also cast into the semantic parsing (He and Golub, 2016; Liang et al., 2017; Yin et al., 2018; Cheng and Lapata, 2018; Guo et al., 2018; Saha et al., 2019). As SQL is used for retrieving information from tables, SPARQL is a popular query language used to retrieve information in a KG. Then naturally, by modifying the NLQ2SQL framework to NLQ2SPARQL, we can tackle the QA over KG using the well-established Seq2Seq based methods. SPARQL queries, which does not involve collecting scattered information (*i.e., JOIN* in SQL), are more concise and shorter than SQL queries for the same natural language questions. This is advantageous for query generation because there is less possibility of error propagation in Seq2Seq framework with autoregressive characteristics. Thus, we assume that SPARQL may be the more suitable logical form than SQL for the NLQ2Query task. To demonstrate this assumption, we construct NLQ-SPARQL pairs and compare the statics of the SPARQL to that of SQL. Actually, as shown in Figure 4, when the number of *JOIN*s increases, which suggests the difficulty of the question, we see that the average length of SPARQL per *JOIN* is similar to that of the natural language question compared to SQL. Furthermore, we evaluate the state-of-the-art EHR QA model TREQS in both NLQ2SQL and NLQ2SPARQL settings where the model showed consistent performance increase in the latter.

3. Dataset

To the best of our knowledge, there is no existing dataset for graph-based EHR QA (*i.e.,* NLQ2SPARQL). In this section, we address the schema issue of `MIMICSQL` and create `MIMICSQL*` that preserves the true structure of MIMIC-III.

3.1. MIMICSQL to MIMICSQL*

Wang et al. chose a subset of information from MIMIC-III; among the total twenty six tables, they chose nine (Patients, Admissions, Diagnoses, Prescriptions, Procedures, Labs, Diagnosis Descrip-
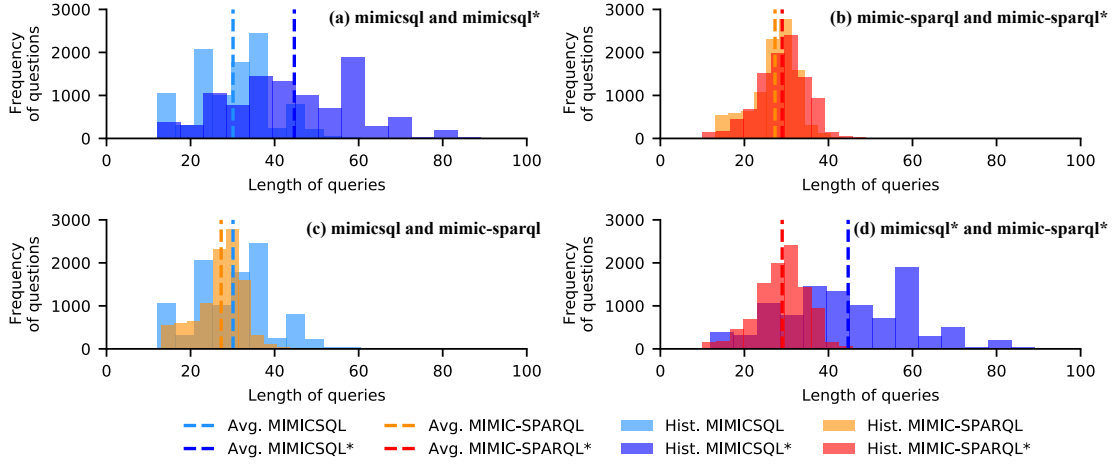
Figure 2: Histograms of query length for `MIMICSQL`, `MIMIC-SPARQL`, `MIMICSQL⋆` and, `MIMIC-SPARQL⋆`. We compared the histograms of each dataset for four cases. (a) and (b): When the schema becomes complex while following the original schema of MIMIC-III, the length of the query in both SQL and SPARQL increases. (c) and (d): When comparing SPARQL and SQL between datasets of the same level, the length of the SQL is increased significantly compared to that of SPARQL.

tions, Procedure Descriptions, and Lab Descriptions). Then they merged those nine tables into five (Demographics, Diagnoses, Procedures, Prescriptions, and Labs) in order to simplify the information structure, which leads to simplified SQL structure. By doing so, they compromised database normalization, such as *JOIN*ing Diagnoses with Diagnosis Descriptions, thus causing duplicate column values in multiple rows. To collect NLQ-SQL pairs, Wang et al. (2020) first define NLQ-SQL templates in the same manner with Pampari et al. (2018); they automatically generate the complete NLQ-SQL pairs (template questions), by randomly sampling the condition values from the five tables. However, these template-based NLQ-SQL pairs have some drawbacks such as not being linguistically diverse as natural questions asked by humans, and being grammatically awkward or inaccurate. To generate more natural questions, the authors refine template questions into natural questions that a medical expert might ask by employing human annotators with clinical knowledge. Consequently, as shown in Table 1, the natural questions has more diverse vocabulary than the template questions, while having a slightly shorter average question length. These statistics indicate that for the semantically same questions, the natural versions are a bit less descriptive than the template versions, while the expression of natural versions is more diverse. Both the template version and the natural version consist of $8,000$ samples for training, $1,000$ for validation, and $1,000$ for testing.

Although `MIMICSQL` can serve as a basic clinical QA dataset, their tables are unnormalized and dissimilar from the tables used in actual hospitals. In other words, a machine trained on `MIMICSQL` would not generalize well to an actual hospital environment, since it was trained in an unrealistic problem space. Therefore, to preserve the schema of MIMIC-III, we normalize `MIMICSQL` tables back to the original nine tables, following the original schema. We also modified the SQL part of `MIMICSQL`'s NLQ-SQL pairs, such that the new tables and columns were correctly reflected. For

example, in `MIMICSQL`, a patient's name and age are stored in the Demographic table. However, the original schema requires a JOIN operation to find a patient's age by their names, since they are stored in two separate tables, namely Admissions and Patients respectively. This modification increased the average number of tokens of SQL from 21.04 to 44.68, due to the added JOIN operations. We denote this modified dataset as `MIMICSQL*`.

3.2. Table to Knowledge Graph

Knowledge Graphs (KG) are a set of triples that describe the relationship between two entities or between an entity and a literal value (Chakraborty et al., 2019). To extract triples from the nine tables of `MIMICSQL*`, we defined a simple algorithm to map each column to either an entity, a literal, or a relation. The primary or foreign key columns of a table are defined as entities, and non-key columns (*i.e.,* property columns) are defined as literals. For example, in the Patients table of Figure 1, the values of SUBJ_ID are defined as entities that can have child nodes. The values of Name, DOB (Date-of-Birth), and Gender are defined as literals that cannot have child nodes. The column names of the non-key columns become relations, *e.g.,* /Name links between SUBJ_ID/12 and "John". Lastly, the relation between a primary key and a foreign key is derived from the table name (*e.g.,* /Admission links between SUBJ_ID/12 and ADM_ID/231). The final KG consists of 173,096 triples and has a max depth of five.
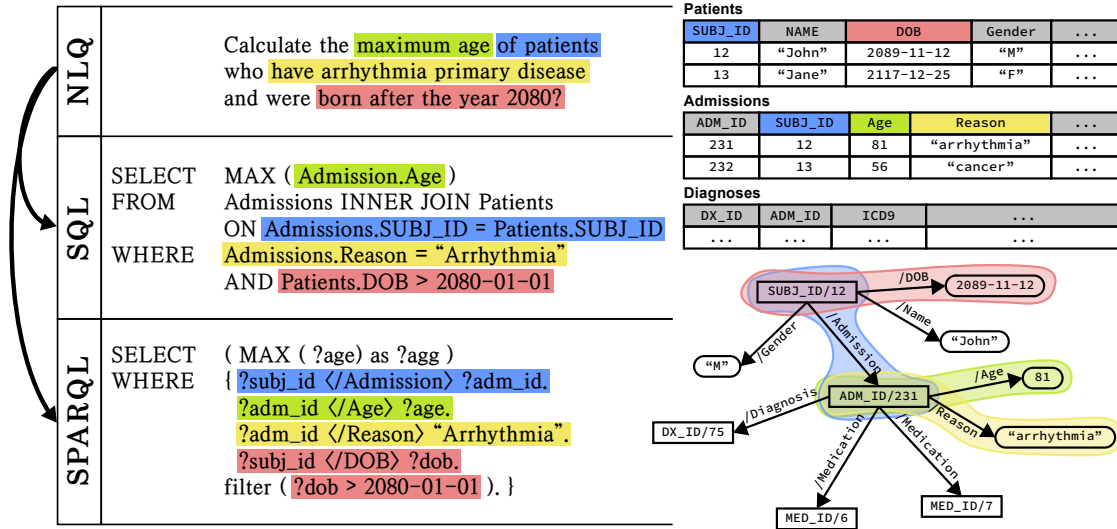

Figure 3: SQL & SPARQL Pair: The highlighted parts in SQL and SPARQL correspond to each other. SQL requires explicit JOIN operations when linking information across tables, whereas SPARQL just hops between nodes.

3.3. SQL to SPARQL

In order to create NLQ-SPARQL pairs from NLQ-SQL pairs, we treat the *SELECT* clause and the *WHERE* clause of SPARQL separately. First, we copy the aggregation function (*e.g.,* MAX, AVERAGE) and the columns from the *SELECT* part of SQL, and fill the SPARQL template. Then we

extract the condition columns, comparison operator, and condition values from the *WHERE* part of SQL. In order to fill the *WHERE* clause of SPARQL, which consists of triples between variables, we find the shortest distance between the variable in the KG[2]. As for combining comparison operators (*e.g.,* $>, <, =$) with triples, equal operators are handled by replacing the variable with values, whereas all other operators are handled by adding filter statements. An example of SQL-to-SPARQL conversion is depicted by Figure 3. After the conversion, we checked the semantic equivalence of SQLs and SPARQLs by querying them to the tables and the KG, where we confirmed a 100% match. The KG and SPARQL queries comprise the graph-based EHR-QA dataset `MIMIC-SPARQL*`. Note that we also constructed `MIMIC-SPARQL` based on the five tables of `MIMICSQL` for further empirical analysis. Table 1 describes the basic statistics of `MIMICSQL*`, `MIMIC-SPARQL*`, `MIMICSQL`, and `MIMIC-SPARQL`.

4. Experiments

This section describes the setting and results of experiments to demonstrate the superior EHR QA performance of the graph-based approach compared to the table-based approach. All datasets and codes will be open-sourced after the review period.

4.1. Experiments Setting

4.1.1 Comparison Methods.   We employed two encoder-decoder based approaches: Seq2Seq model (Luong et al., 2015) and TREQS (Wang et al., 2020). Seq2Seq uses a bidirectional LSTM encoder and an LSTM decoder with an attention mechanism, but it cannot handle out-of-vocabulary (OOV) tokens that frequently occur in medical data. Proposed with `MIMICSQL`, TREQS is a state-of-the-art NLQ2SQL translation model for clinical questions, equipped with the copying mechanism (See et al., 2017) to address the OOV issue, and the attention mechanism (Bahdanau et al., 2015) to help SQL token generation.

The LSTM-based Seq2Seq model was implemented with PyTorch. As for TREQS, we used the official code released by the original authors, rather than implementing from scratch. In order to carefully compare the grammatical difference between two query languages without modifying the model architecture, we utilized the official code[3] of TREQS and hyperparameters described in (Wang et al., 2020). We only tuned hyperparameters that are irrelevant to the model size. The number of trainable parameters of TREQS is $2,795,907$ and Seq2Seq is $3,844,096$ throughout all experiments. As shown in Table 4, the hyperparameters are chosen to optimize the model for the dataset without increasing the model size and sampled from uniform distribution. We train all models from scratch to 20 epochs. The development set is utilized to select the best configuration. In the test time, we employ a beam search algorithm to generate the output query sentences where the beam size is set to 5. Training both models required an average of one hour on a TITAN-Xp GPU.

4.1.2 Dataset Descriptions.   The datasets used in the experiment are grouped according to three criteria. First, the QA datasets are categorized into NLQ-SQL pairs (`MIMICSQL` and `MIMICSQL*`) and NLQ-SPARQL pairs (`MIMIC-SPARQL` and `MIMIC-SPARQL*`) depending on the type of a

---

2. Since we know the schema of MIMIC-III, we can create a relation graph and apply Dijkstra's algorithm.

3. https://github.com/wangpinggl/TREQS

logical form. Note that NLQs are the same for all datasets, and therefore any performance difference is solely caused by whether we use the graph-based or table-based approach. The second criterion is whether to follow the original schema of MIMIC-III. `MIMICSQL` is constructed with a simplified MIMIC-III schema, whereas `MIMICSQL*` maintains the original schema of MIMIC-III. Since the knowledge graph is created by converting the MIMIC-III tables, `MIMIC-SPARQL` and `MIMIC-SPARQL*` were converted from their respective counterparts, `MIMICSQL` and `MIMICSQL*`. Lastly, there are two versions of NLQ, namely template questions and natural language questions. As described in Section 3.1, the NLQ-SQL pairs in MIMICSQL were constructed based on human-defined SQL templates. As seen in Table 1, the template questions are less diverse compared to the natural questions in terms of linguistic expression, making it easier for the model to understand the intent of the question. While natural questions are more similar to human language than template questions are, it is generally more difficult for the model to understand the intent.

To demonstrate the performance between SPARQL and SQL, we compare the performance of models trained with `MIMICSQL` against `MIMIC-SPARQL`, and `MIMICSQL*` against `MIMIC-SPARQL*`. Additionally, we found an improved method to tokenize SQLs of `MIMICSQL`. In the SQL queries provided in the original `MIMICSQL`, the table name and the column name are not separated and are considered as a single token. This makes it difficult for the model to learn the relationship between the *SELECT* clause and *FROM* clause. Thus, we separate the table name from the column name. The effectiveness of this tokenization is explained in Section 4.2.

4.1.3 Evaluation Metrics. We use the following three metrics to measure the question answering performance of the models. Table 2 shows the ground truth SQL and four SQL examples to elaborate how each metric is measured.

- **Logic Form Accuracy (Zhong et al., 2017)** (Acc_{LF}) is calculated by comparing the generated SQL/SPARQL queries with the true SQL/SPARQL queries token-by-token. This is the most strict way of measuring the model performance.

- **Execution Accuracy** (Acc_{EX}) is calculated based on the correctness of the answer retrieved by querying the tables/KG with the generated SQL/SPARQL. This is a more lenient way of measuring the model performance, since the model can generate incorrect SQL/SPARQL queries but still get the correct answer by luck.

- **Structural Accuracy** (Acc_{ST}) is similar to Acc_{LF}, but disregards the condition value tokens (*e.g.,* numeric values or string values). We use Acc_{ST} to evaluate how well the model understands the relations between columns/entities, and successfully converts NLQ to SQL/SPARQL structure, with less emphasis on the exact condition values (similar to how models are evaluated for the Spider dataset (Yu et al., 2018)).

4.2. Experiments Results

4.2.1 SQL vs SPARQL. As shown in Table 3, regardless of the type of NLQ (template or natural), the SQL dataset (`MIMICSQL`) shows competitive performance to its SPARQL counterpart (`MIMIC-SPARQL`) when using the simplified schema (*i.e.,* five tables). When using the original MIMIC-III schema, however, the graph-based approach (`MIMIC-SPARQL*`) consistently outperforms the table-based approach (`MIMICSQL*`). It is noteworthy that the Acc_{ST} gap between SQL

Table 2: Four pseudo-SQL examples of measuring Acc_{LF}, Acc_{EX}, and Acc_{ST}

Ground Truth	*select max(age) from patients where Gender = "F" and DoB > 2020*	LF	EX	ST
Predicted SQL1	*select max(age) from patients where Gender = "F" and DoB > 2020*	✓	✓	✓
Predicted SQL2	*select max(age) from patients where DoB > 2020 and Gender = "F"*	✗	✓	✓
Predicted SQL3	*select max(age) from patients where DoB > 2021 and Gender = "M"*	✗	✗	✓
Predicted SQL4	*select max(age) from patients where DoB > 2021 and Diagnosis = "F"*	✗	✗	✗

Table 3: Medical QA performance on template (Top) and natural (Bottom) questions evaluated with logic form accuracy (Acc_{LF}), execution accuracy (Acc_{EX}) and the structural accuracy (Acc_{ST}). In addition to MIMICSQL, we include MIMICSQL (Paper), the performance reported in Wang et al. (2020) and explained in Appendix .1

Method	Dataset Template Question	# of Tables	Development			Testing		
			Acc_{LF}	Acc_{EX}	Acc_{ST}	Acc_{LF}	Acc_{EX}	Acc_{ST}
Seq2Seq	MIMICSQL (Paper)		0.098	0.372	-	0.160	0.323	-
	MIMICSQL	5	0.327 (±0.118)	**0.429** (±0.075)	0.467 (±0.028)	0.410 (±0.120)	0.455 (±0.083)	0.546 (±0.022)
	MIMIC-SPARQL		**0.337** (±0.117)	0.383 (±0.073)	**0.877** (±0.029)	**0.426** (±0.120)	**0.463** (±0.089)	**0.892** (±0.016)
	MIMICSQL⋆	9	0.298 (±0.156)	**0.407** (±0.099)	0.455 (±0.017)	0.348 (±0.158)	0.418 (±0.104)	0.512 (±0.020)
	MIMIC-SPARQL⋆		**0.333** (±0.134)	0.380 (±0.088)	**0.874** (±0.022)	**0.415** (±0.127)	**0.461** (±0.086)	**0.858** (±0.015)
TREQS	MIMICSQL (Paper)		0.712	0.803	-	0.802	0.825	-
	MIMICSQL	5	**0.752** (±0.025)	0.810 (±0.012)	0.982 (±0.057)	**0.823** (±0.038)	**0.843** (±0.038)	0.986 (±0.082)
	MIMIC-SPARQL		0.735 (±0.017)	**0.816** (±0.014)	**0.989** (±0.008)	0.805 (±0.010)	0.841 (±0.012)	**0.990** (±0.007)
	MIMICSQL⋆	9	0.672 (±0.053)	0.789 (±0.027)	0.921 (±0.011)	0.749 (±0.047)	0.816 (±0.032)	0.952 (±0.015)
	MIMIC-SPARQL⋆		**0.745** (±0.007)	**0.824** (±0.007)	**0.994** (±0.001)	**0.814** (±0.010)	**0.848** (±0.007)	**0.991** (±0.006)

Method	Dataset Natural Question	# of Tables	Development			Testing		
			Acc_{LF}	Acc_{EX}	Acc_{ST}	Acc_{LF}	Acc_{EX}	Acc_{ST}
Seq2Seq	MIMICSQL (Paper)		0.076	0.112	-	0.091	0.131	-
	MIMICSQL	5	**0.197** (±0.031)	**0.335** (±0.024)	0.360 (±0.026)	**0.261** (±0.031)	**0.358** (±0.030)	0.404 (±0.025)
	MIMIC-SPARQL		0.179 (±0.093)	0.272 (±0.056)	**0.637** (±0.019)	0.252 (±0.099)	0.338 (±0.065)	**0.634** (±0.016)
	MIMICSQL⋆	9	0.148 (±0.040)	**0.301** (±0.014)	0.327 (±0.012)	0.181 (±0.040)	0.293 (±0.030)	0.347 (±0.018)
	MIMIC-SPARQL⋆		**0.185** (±0.132)	0.285 (±0.092)	**0.623** (±0.039)	**0.225** (±0.133)	**0.321** (±0.094)	**0.590** (±0.038)
TREQS	MIMICSQL (Paper)		0.451	0.511	-	0.486	0.556	-
	MIMICSQL	5	**0.588** (±0.005)	0.707 (±0.009)	0.825 (±0.005)	**0.661** (±0.030)	**0.732** (±0.024)	**0.821** (±0.023)
	MIMIC-SPARQL		0.585 (±0.006)	**0.721** (±0.006)	**0.828** (±0.008)	0.649 (±0.016)	0.729 (±0.012)	0.818 (±0.016)
	MIMICSQL⋆	9	0.530 (±0.031)	0.689 (±0.019)	0.736 (±0.030)	0.601 (±0.031)	0.694 (±0.020)	0.743 (±0.036)
	MIMIC-SPARQL⋆		**0.577** (±0.017)	**0.715** (±0.013)	**0.818** (±0.011)	**0.633** (±0.010)	**0.722** (±0.006)	**0.788** (±0.010)

and SPARQL dramatically increases when moving from the modified schema to the original, indicating that the models better understand the complex relations between pieces of information when trained on the SPARQL dataset. Furthermore, while there was a significant performance drop when moving from MIMICSQL to MIMICSQL⋆, especially when using a simpler model (*i.e.,* Seq2Seq), there was either marginal decrease or even slight increase in performance when moving from MIMIC-SPARQL to MIMIC-SPARQL⋆, indicating the effectiveness of the graph-based approach as the knowledge structure becomes more complex.

These results support our hypothesis that the graph-based approach could be more suitable for EHR QA as graphs can represent relations between entities and values more concisely compared to tables, which require *JOIN* operations. Specifically, the statistics of the dataset in Table 1 show that the query length of MIMICSQL⋆ is nearly twice as long as that of MIMIC-SPARQL⋆. This

is because a single *JOIN* in SQL requires 11 tokens (including '=' and '.'), while a single hop in SPARQL requires only 3 (*i.e.,* subject, predicate, and object). This characteristic between SQL and SPARQL is also explicitly visualized in Figure 2, where we can readily observe the drastic change of SQL query length when moving from simple to complex schema, whereas SPARQL query length stays relatively constant. Additionally, unlike SPARQL, SQL requires the model to learn the hierarchy between a table and its columns, in addition to the relations between tables. Such syntactic difference between SQL and SPARQL originates from the intrinsic difference between the relational tables and a knowledge graph in terms of how to connect information (joining multiple tables versus hopping across triples). This leads to the superior performance of the graph-based approach over the table-based one especially in Acc_{ST}, which only evaluates the structural accuracy (*i.e.,* disregarding the correctness of the condition values).
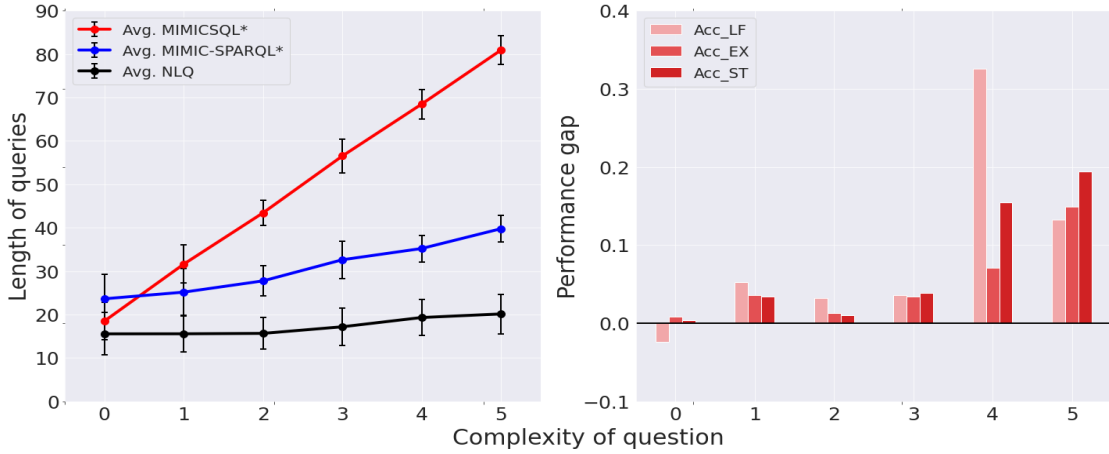

Figure 4: *Left:* When the complexity of the queries increases, the length of all queries increases. The length of SPARQLs increases with a slope more similar to that of NLQs than that of SQL queries. *Right:* This leads to the increased performance gap between graph-based and table-based models for complex questions.

4.2.2 Analysis of NLQ type: Template and Natural. As mentioned in Section 3.1, MIMICSQL provides both template questions and natural questions, thus giving us the option to either convert template questions to SQL/SPARQL, or convert natural questions to SQL/SPARQL. As the vocabulary sizes in Table 1 indicates, the linguistic patterns of template questions are less diverse than the patterns of natural questions because template questions are generated by sampling only the condition values based on a predefined template to create the NLQ2Query pairs. On the other hand, the natural questions contain various phrases thanks to the refinement process by human annotators. Naturally, the more diverse the question patterns are, the harder the task is for the model to correctly encode the given questions to a latent space. As shown in Table 3, training the model on template questions demonstrates superior performance compared to training the model on natural questions, regardless of whether we use the table-based approach or the graph-based approach. It is noteworthy that the graph-based approach demonstrated approximately 30% higher Acc_{ST} than the table-based approach. Moreover, in the natural questions, regardless of the logical form and model, all models lose performance when the schema becomes complex from five tables to nine tables. However, in

the template questions, TREQS trained with SPARQL maintains the performance even if the schema becomes complex. This indicates that the graph-based approach is robust to the complexity of the schema when the questions are straightforward enough to encode (*i.e.,* less diverse vocabulary) as explained in Section 3.1.

NLQ	how many of the patients had other shock without mention of trauma?
GT **MIMICSQL***	select count (distinct patients . "subject_id") from patients inner join admissions on patients . "subject_id" = admissions . "subject_id" inner join diagnoses on admissions . "hadm_id" = diagnoses . "hadm_id" inner join d_icd_diagnoses on diagnoses . "icd9_code" = d_icd_diagnoses . "icd9_code" where d_icd_diagnoses . "long_title" = "other shock without mention of trauma"
Seq2Seq **MIMICSQL***	select count (distinct patients . "subject_id") from patients inner join admissions on patients . "subject_id" = admissions . "subject_id" inner join diagnoses on admissions . "hadm_id" = diagnoses . "hadm_id" inner join d_icd_diagnoses on diagnoses . "icd9_code" = d_icd_procedures . "icd9_code" where d_icd_diagnoses . "long_title" = \<UNK\> of \<UNK\> of \<UNK\> and \<UNK\>
TREQS **MIMICSQL***	select count (distinct patients . "subject_id") from patients inner join admissions on patients . "subject_id" = admissions . "subject_id" inner join procedures on admissions . "hadm_id" = procedures . "hadm_id" inner join d_icd_procedures on procedures . "icd9_code" = d_icd_procedures . "icd9_code" where d_icd_procedures . "long_title" = "other shock without mention of trauma"
GT **MIMIC–** **SPARQL***	select (count (distinct ?subject_id) as ?agg) where { ?subject_id \</hadm_id\> ?hadm_id. ?hadm_id \</diagnoses\> ?diagnoses. ?diagnoses \</diagnoses_icd9_code\> ?diagnoses_icd9_code. ?diagnoses_icd9_code \</diagnoses_long_title\> "other shock without mention of trauma" ^^\<http://www.w3.org/2001/xmlschema#string\>. }
Seq2Seq **MIMIC–** **SPARQL***	select (count (distinct ?subject_id) as ?agg) where { ?subject_id \</hadm_id\> ?hadm_id. ?hadm_id \</diagnoses\> ?diagnoses. ?diagnoses \</diagnoses_icd9_code\> ?diagnoses_icd9_code. ?diagnoses_icd9_code \</diagnoses_long_title\> "other other shock mention of \<UNK\>" ^^\<http://www.w3.org/2001/xmlschema#string\>.
TREQS **MIMIC–** **SPARQL***	select (count (distinct ?subject_id) as ?agg) where { ?subject_id \</hadm_id\> ?hadm_id. ?hadm_id \</diagnoses\> ?diagnoses. ?diagnoses \</diagnoses_icd9_code\> ?diagnoses_icd9_code. ?diagnoses_icd9_code \</diagnoses_long_title\> "other shock without mention of trauma" ^^\<http://www.w3.org/2001/xmlschema#string\>. }

Figure 5: TREQS learned with `MIMIC-SPARQL*` correctly generated query structure and condition values. However, even though it is the same architecture, when learned with `MIMICSQL*`, the model incorrectly generated the structure of SQL (highlighted in red).

4.2.3 Performance Gap by Question Complexity. We can reliably say that if an SQL query contains many *JOIN* operations, then the corresponding NLQ involves multiple logical connections (*i.e.,* reasoning) to find the answer. Thus, we can safely assume that the complexity of the NLQ is proportional to the number of *JOIN*s. As explained in Section 3.3, the SPARQL datatsets share the same NLQ as the SQL datasets, and we can classify the SPARQL queries based on the number of *JOIN*s in the corresponding SQL queries. In addition, from the SPARQL perspective, the number of hops between triples increases when the number of *JOIN*s increases.

We confirmed in Table 3 that the performance drop of table-based approaches was greater than that of graph-based when the schema became complex from the five tables to nine tables. Figure 4 presents possible explanations for this phenomenon. We classified the questions according to complexity, and plotted the performance gap of the two models (*i.e.* TREQS models trained with `MIMICSQL*` and `MIMIC-SPARQL*` respectively), for each question group as a bar graph for the three metrics (see *Right*). The performance gap between two models increases for complex questions that require more than three *JOIN*s. We assumed the different natures of SQL and SPARQL led to this performance gap, and analyzed the sequence length of SQL and SPARQL queries per

varying question complexity. As the complexity of question increases, the length of all types of queries tends to increase (see *Left*). Note that the increase in the length of NLQs is not larger than that of SQL queries. This is because of the syntactic nature of SQL, where the multiple keywords for performing *JOIN*s between tables are added. On the other hand, the SPARQL queries do not increase in length significantly compared to the SQL queries. Through this trend, we judge that the query structure of SPARQL is closer to NLQ than SQL.

4.2.4 Qualitative Comparison between Generated Queries. We demonstrate the qualitative results to study the differences between the model, and the differences between the table-based and the graph-based datasets. For the same NLQ, we present queries for six cases as shown in Figure 5. In terms of the model, TREQS handles OOV correctly by copying the condition value in the NLQ. However, Seq2Seq generates <UNK> tokens for the condition values in both `MIMICSQL*` and `MIMIC-SPARQL*`, since there exist several abbreviations and rarely occurring words in the EHR. In terms of a logical form, the models that learned SQL incorrectly create "*d_icd_diagnoses*", which correspond to the query structure, as "*d_icd_procedures*". On the other hand, although the Seq2Seq model generated condition values incorrectly due to the OOV problem, the models that learned SPARQL generate the query structure perfectly.

5. Discussion & Limitation

From the vast amount of EHR data, retrieving desired information is laborious for both medical experts and ordinary users. EHR QA is a meaningful task in that it provides a natural language interface that allows users to readily extract information. In addition, it is a promising research field that can serve telemedicine to patients in the era where the contact-free treatment becomes important. In this work, we focused on the differences in the data structure of the relational database and the knowledge graph, and analyzed the variation of logical forms in complex schema. Motivated by these findings, we hypothesized that a graph-based approach is a more suitable choice for conducting complex EHR QA than a table-based approach. With extensive experiments, we empirically demonstrated the superior performance of the graph-based approach. To encourage EHR QA research in both directions, we constructed a pair of EHR QA datasets, one table-based (`MIMICSQL*`) and another graph-based (`MIMIC-SPARQL*`). To the best of our knowledge, this is the first work to propose and successfully demonstrate the graph-based EHR QA.

Limitations The graph-based approach, however, has its own limitation in terms of inference time compared to the table-based approach. Note that the SQL processing time scales favorably as the table size grows, compared to the SPARQL processing time as the graph size grows (especially in terms of the number of edges). In this work, the average response time for both the table-based (SQL) and the graph-based (SPARQL) approaches were less than 1 second, since both datasets were constructed with a subset of MIMIC-III. But if we were to convert the entire MIMIC-III dataset to a knowledge graph, then scalability might become a bottleneck when deploying the model in practice (*e.g.* SPARQL taking several seconds to retrieve an answer). Note that, however, this is strictly related to the inference phase, and not the training phase. The model training is actually faster with SPARQL, since SPARQL queries are shorter than SQL queries on average. As future work, we plan to extend both our datasets (`MIMICSQL*` and `MIMIC-SPARQL*`) to cover a larger subset of MIMIC-III and to address the scalability of the graph-based approach.

Acknowledgements

References

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proc. of the IEEE international conference on computer vision (ICCV)*, 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proc. the International Conference on Learning Representations (ICLR)*, 2015.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, oct 2014.

Nilesh Chakraborty, Denis Lukovnikov, Gaurav Maheshwari, Priyansh Trivedi, Jens Lehmann, and Asja Fischer. Introduction to neural network based approaches for question answering over knowledge graphs. *CoRR*, 2019.

Jianpeng Cheng and Mirella Lapata. Weakly-supervised neural semantic parsing with a generative ranker. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, 2018.

Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proc. the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-sql evaluation methodology. In *Proc. the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. Dialog-to-action: Conversational question answering over a large-scale knowledge base. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

Xiaodong He and David Golub. Character-level question answering with attention. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

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

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proc. the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018.

Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. In *Proc. the International Conference on Learning Representations (ICLR)*, 2015.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: a human-generated machine reading comprehension dataset. In *CoCo@ NIPS*, 2016.

Anusri Pampari, Preethi Raghavan, Jennifer Liang, and Jian Peng. emrqa: A large corpus for question answering on electronic medical records. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

Amrita Saha, Ghulam Ahmed Ansari, Abhishek Laddha, Karthik Sankaranarayanan, and Soumen Chakrabarti. Complex program induction for querying knowledge bases in the absence of gold programs. *Transactions of the Association for Computational Linguistics*, 2019.

Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. In *Proc. the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *Proc. the International Conference on Learning Representations (ICLR)*, 2017.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

Ping Wang, Tian Shi, and Chandan K Reddy. Text-to-sql generation for question answering on electronic medical records. In *Proc. the International Conference on World Wide Web (WWW)*, 2020.

Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proc. the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. Structvae: Tree-structured latent variable models for semi-supervised semantic parsing. In *Proc. the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

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

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

Table 4: Description of hyperparameters

Method	Dataset	Hyperparameters		
		Parameter	Search space	Selected value
Seq2Seq	MIMICSQL⋆	Batch Size	$16, 32, 48$	32
		Step decay	$[0.01, 0.8]$	0.1
		Step size	$1, 2, 5, 10$	2
		Learning rate	$[1 \times 10^{-5}, 1 \times 10^{-2}]$	0.001
	MIMIC-SPARQL⋆	Batch Size	$16, 32, 48$	16
		Step decay	$[0.01, 0.8]$	0.8
		Step size	$1, 2, 5, 10$	2
		Learning rate	$[1 \times 10^{-5}, 1 \times 10^{-2}]$	0.001
TREQS	MIMICSQL⋆	Batch Size	$16, 32, 48$	16
		Step decay	$[0.01, 0.8]$	0.8
		Step size	$1, 2, 5, 10$	2
		Learning rate	$[1 \times 10^{-5}, 1 \times 10^{-2}]$	0.0005
	MIMIC-SPARQL⋆	Batch Size	$16, 32, 48$	48
		Step decay	$[0.01, 0.8]$	0.1
		Step size	$1, 2, 5, 10$	2
		Learning rate	$[1 \times 10^{-5}, 1 \times 10^{-2}]$	0.0005

Appendix A.

.1. Performance Gap between `MIMICSQL` and `MIMICSQL` (paper)

As shown in bottom Table 3, there is performance gap between the reproduced results and the reported results in original paper (Wang et al., 2020). In the queries provided by the original MIMICSQL, the table name and the column name are treated as a single token when being fed to the model. This makes it difficult for the model to learn the relationship between the *SELECT* clause and *FROM* clause. Thus, we separate the table name from the column name and report the effectiveness of this tokenization. Table 3 shows that training models on MIMICSQL increases performance for all metrics compared to training models on MIMICSQL (paper). By simply changing the tokenization method, the execution accuracy increased by 22% and 19% in the Seq2Seq and TREQS that are trained with the natural questions, respectively.