Question Answering for Complex Electronic Health Records Database using Unified Encoder-Decoder Architecture

Seongsu Bae   SEONGSU@KAIST.AC.KR
KAIST AI / Daejeon, South Korea

Daeyoung Kim   DAEYOUNG.K@KAIST.AC.KR
KAIST AI / Daejeon, South Korea

Jiho Kim   JIHO283@KAIST.AC.KR
KAIST AI / Daejeon, South Korea

Edward Choi   EDWARDCHOI@KAIST.AC.KR
KAIST AI / Daejeon, South Korea

Abstract

An intelligent machine that can answer human questions based on electronic health records (EHR-QA) has a great practical value, such as supporting clinical decisions, managing hospital administration, and medical chatbots. Previous table-based QA studies focusing on translating natural questions into table queries (NLQ2SQL), however, suffer from the unique nature of EHR data due to complex and specialized medical terminology, hence increased decoding difficulty. In this paper, we design UniQA, a unified encoder-decoder architecture for EHR-QA where natural language questions are converted to queries such as SQL or SPARQL. We also propose input masking (IM), a simple and effective method to cope with complex medical terms and various typos and better learn the SQL/SPARQL syntax. Combining the unified architecture with an effective auxiliary training objective, UniQA demonstrated a significant performance improvement against the previous state-of-the-art model for MIMICSQL* (14.2% gain), the most complex NLQ2SQL dataset in the EHR domain, and its typo-ridden versions ($\approx 28.8\%$ gain). In addition, we confirmed consistent results for the graph-based EHR-QA dataset, MIMICSPARQL*.

Keywords: Electronic Health Records, Natural Language Processing

1. Introduction

Electronic health records (EHR) consist of real-world clinical data (e.g., patient diagnoses, medications,

Figure 1: An example of natural language question (NLQ) and SQL query pairs for EHR question answering.

lab results), usually stored in a complex relational database (RDB) such as MIMIC-III (Johnson et al.,

2016) and eICU (Pollard et al., 2018). For healthcare providers or ordinary users, performing information retrieval or knowledge inference from such massive hospital database systems is not a trivial task. They not only must learn to use an appropriate query language (e.g., SQL, SPARQL) but also learn the schema and the corresponding values in the hospital RDB. Therefore developing an intelligent EHR Question Answering (EHR-QA) model that can answer human-level questions from the database has immense practical values such as supporting clinical decisions, managing hospital administration, and medical chatbots.

Recently, translating natural language questions into corresponding SQL queries (*i.e.* NLQ2SQL) (Zhong et al., 2017; Hwang et al., 2019; Wang et al., 2020a) has become the dominant approach in QA over relational databases. In the healthcare domain, Wang et al. (2020b) was the first attempt to construct MIMICSQL, a large-scale NLQ2SQL dataset built from the open-source EHR dataset MIMIC-III (Johnson et al., 2016), which was followed by MIMICSQL* and MIMICSPARQL* (Park et al., 2021), more refined EHR-QA datasets derived from MIMICSQL.

Unlike the general domain QA, EHR-QA faces a unique challenge due to the distinguished nature of EHR data. Figure 1 shows an illustrative example of EHR-QA, which presents a pair of the natural language question and the corresponding SQL query and multiple relational tables required to retrieve it. Although the input question (*i.e.* "*How many patients had the diagnosis...?*") is of natural syntax, it contains long and domain-specific terms, such as "*hemochromatosis due to repeated red blood cell transfusions*", which can lead to increased decoding difficulty and various typos, such as missing or reversed letters.

In this work, we propose UniQA, an effective EHR-QA model based on the unified encoder-decoder architecture to cope with the discussed practical and realistic challenge. Using a combination of the unified *Encoder-as-Decoder* architecture, input token masking, and the value recovering technique, UniQA was able to achieve the state-of-the-art EHR-QA performance on MIMICSQL* as well as robustness against its variants with various typos for input questions. Furthermore, UniQA showed consistent empirical results for the graph-based EHR-QA dataset, MIMIC-SPARQL*.

The contributions of this work can be summarized as follows:

- We propose UniQA, a unified *Encoder-as-Decoder* model to answer EHR-related questions, achieving the state-of-the-art performance (14.2% improvement over the previous SOTA) on the latest EHR-QA dataset (*i.e.* MIMIC-SQL*).

- We propose a simple and effective training objective, Input-Masking (IM) which is an effective solution to cope with various input typos. We demonstrate that our masking strategy yields about 28.8% improvement over the previous state-of-the-art model for typo-ridden MIMICSQL*.

- We further conducted a comprehensive analysis and confirmed the efficacy of UniQA, especially verifying the consistent results on the latest graph-based EHR-QA dataset (*i.e.* MIMIC-SPARQL*)

2. Related Work

2.1. NLQ to Query Language Generation

For question answering over relational databases, translating natural language questions into corresponding queries (NLQ2Query) has become the dominant approach. Existing NLQ2Query datasets and approaches can be categorized depending on their main purposes: generalization over cross domains or specialization within a target domain. For the former case, a variety of methods (Guo et al., 2019; Choi et al., 2020; Wang et al., 2020a) were designed to handle unseen queries or databases during evaluation, mainly based on WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018). Those models are more focused on inferring complex query structures (*i.e.* SQL syntax, tables, and column names) rather than parsing and predicting desired condition values related to the validity of final execution. On the other hand, domain-specific datasets (Price, 1990; Quirk et al., 2015; Li and Jagadish, 2014) have been studied for a longer period of time (Giordani and Moschitti, 2012; Dong and Lapata, 2018). Most of datasets, however, are small (< 1,000 samples) and are used as another sources to measure generalization. For the healthcare domain, there are two publicly available NLQ2Query datasets over EHR database (Wang et al., 2020b; Park et al., 2021) in terms of viewing EHR as relational tables or a massive knowledge graph.

2.2. Electronic Health Records QA

Recent EHR-QA research can be classified into two main categories: unstructured QA and structured QA. For the former case, QA research has been mainly developed as machine reading comprehension which extracts the answer to the given question from free text such as clinical case reports (Šuster and Daelemans, 2018), clinical notes (Pampari et al., 2018), and healthcare articles (Zhu et al., 2020).

For the structured case, it can be divided into table-based QA and graph-based QA according to the structure of the knowledge base. TREQS (Wang et al., 2020b) is the table-based QA model solving the NLQ2SQL task over MIMICSQL which is an EHR-QA dataset derived from MIMIC-III (Johnson et al., 2016), an open-source dataset for ICU records. Raghavan et al. (2021) also proposed emrKBQA, but not publicly available yet, another table-based QA dataset aimed at semantic parsing to map natural language questions to logical forms from the structural part of the EHR (*i.e.* MIMIC-III). For the counterpart of table-based QA, Park et al. (2021) extended the field of structured EHR-QA by transforming MIMICSQL's tables to a knowledge graph, thus proposing a graph-based EHR-QA. Furthermore, Park et al. (2021) empirically demonstrated that a graph-based approach is more suitable for conducting complex EHR-QA than a table-based approach. In all previous works, however, the EHR-QA task was tackled with the classical encoder-to-decoder architecture implemented with RNNs, without considering the complex nature of EHR and the practical challenges induced by it.

3. Method

3.1. Problem Setup

Our goal is to transform the natural questions asked by the user into executable queries (*i.e.* SQL or SPARQL). For notation, we define a natural language question Q as a series of n tokens (*i.e.* subwords), $Q = \{q_1, q_2, \ldots, q_n\}$. Similarly, we define Y as the corresponding query consisting of m tokens, $Y = \{y_1, y_2, \ldots, y_m\}$. The goal of our model is to maximize the probability $P(Y|Q)$. Note that similar to Wang et al. (2020b) and Park et al. (2021), we do not rely on the meta information of the knowledge source (*e.g.* database schema or knowledge graph structure) in order to improve the generality of our approach (*i.e.* usable for both NLQ2SQL and NLQ2SPARQL). Instead, we assume that the meta
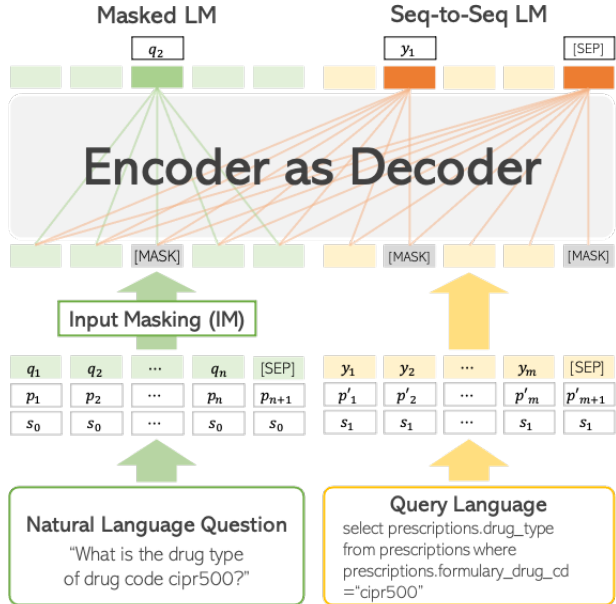

Figure 2: Overview of training our UniQA model. The input q_i and y_j are summed with the corresponding segment embedding $s_{0/1}$ and the position embedding p_i, p'_j, respectively. Two training objectives are used: Masked LM for the question part and seq-to-seq LM for the query language part. The colored line represents whether a pair of tokens can be attended to each other.

information is implicitly expressed by the natural language questions, and let the model learn it via the training process.

3.2. Input Representation

Let the model input be a sequence which consists of two sub-sequences: the natural language question $Q = \{q_1, q_2, \ldots, q_n\}$ and corresponding query $Y = \{y_1, y_2, \ldots, y_m\}$. Input questions and queries consist of multiple tokens, tokenized by subword units by WordPiece (Wu et al., 2016) regardless of subsequence types. Both Q and Y are converted to a sequence of token embeddings $\{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n\}$ and $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_m\}$ via a trainable lookup table, resepectively. Then we place a special separator token embedding $[SEP]$ each at the end of Q and Y to form a model input as shown in Figure 2. Then each token embedding is summed with the corresponding position embedding $\boldsymbol{p}$ and segment embedding $\boldsymbol{s}$ to prepare the final input to the model.

3.3. Encoder-as-Decoder Architecture

Inspired by Dong et al. (2019), we propose an *Encoder-as-Decoder* model suited for the NLQ2Query task. To the best of our knowledge, this is the first attempt to adapt the encoder-as-decoder framework into the NLQ2Query task. Denoting the input embeddings from Section 3.2 as $\boldsymbol{H}^0$, they are encoded into contextual representations at different levels of hidden outputs $\boldsymbol{H}^l$ using an L-layer Transformer encoders $\boldsymbol{H}^l = \text{Transformer}(\boldsymbol{H}^{l-1})$, $l \in [1, L]$ (Vaswani et al., 2017).

In this unified architecture, decoding is performed similar to Dong et al. (2019); At inference, given the input sequence $(Q, [\text{SEP}], [\text{MASK}]_1)$, the model predicts $\hat{y}_1$, the identity of $[\text{MASK}]_1$. Then $[\text{MASK}]_1$ is replaced with $\hat{y}_1$ and we attach $[\text{MASK}]_2$ to the previous input sequence and repeat this process until [SEP] token is predicted.

In a typical *Encoder-to-Decoder* architecture, the decoder can only access the fully contextualized input embeddings (*i.e.* the output of the encoder), thereby limiting the model's ability to consider the input tokens during the decoding process. On the other hand, the *Encoder-as-Decoder* architecture allows the decoding process to access the input tokens at every layer of the encoder, thus improving the decoding capacity. Moreover, thanks to both encoding and decoding trained with [MASK] reconstruction (unlike Encoder-to-Decoder where the decoder is trained autoregressively), Encoder-as-Decoder can be naturally initialized with pre-trained language models such as BERT (Devlin et al., 2019).

3.4. Input Masking on NLQ

Natural questions in the EHR-QA are prone to various typos such as reversed or missing letters, due to the complex and specialized medical terms. In order to make the encoder-as-decoder model more robust to various typos, we use an additional training strategy named *Input Masking*. During training, tokens in the input NLQ $\{q_1, q_2, \ldots, q_n\}$ are randomly masked with probability 0.2. We replace the chosen token with the [MASK] token with 80% chance, a random token with 10% chance, and the original token with 10% chance. This technique, which can be seen as a form of data augmentation, encourages the model to rely more on familiar tokens when unfamiliar tokens (*i.e.* typos) are included in the input.

In addition, Input Masking can help the model better learn the syntactic structure of SQL (or SPARQL)

queries. For example, typos can occur at various places other than medical terms in the NLQ (*i.e.* retrieve can be misspelled as retreive), which can obstruct accurate decoding. IM, however, can alleviate this type of challenge as it will make the model robust to any kind of typos in the NLQ. We empirically demonstrate the effectiveness of Input Masking in the experiments by evaluating the proposed training strategy on datasets with different levels of noise (*i.e.* typos).

3.5. Model Training

We initialize the encoder-as-decoder model with BERT-base (Devlin et al., 2019) and train our model with the two training objectives: 1) Masked Language Modeling (MLM) is used for reconstructing the input masks applied to the NLQ part; 2) Sequence-to-Sequence (Seq2Seq) is used to train our model to act as a decoder. The difference between MLM and Seq2Seq is the structure of the attention masks. As shown in Fig. 2, the NLQ tokens can freely attend to one another but not to the query tokens. The query tokens, on the other hand, can only attend to the previous query tokens but freely attend to all NLQ tokens. By jointly optimizing with two aforementioned training objectives, the model should recover for masked NLQ tokens while also predicting the corresponding query tokens.

4. Experiments

4.1. Experiment Settings

4.1.1. DATASET

MIMICSQL* For evaluating the models, we use MIMICSQL* (Park et al., 2021), which is a table-based EHR-QA dataset consisting of 10,000 NLQ-SQL pairs derived from 9 tables[1] of MIMIC-III (Johnson et al., 2016), an open-source ICU dataset. For all experiments, we use 8,000 pairs for training, 1,000 for validation and 1,000 for testing. Note that MIMICSQL* is a revised version of MIMISQL (Wang et al., 2020b), where the authors restored the original MIMIC-III schema as well as improved the tokenization of both NLQ and SQL.

NOISY MIMICSQL* As shown in Figure 3, MIMICSQL* has longer input questions on average compared to existing NLQ2Query datasets, which can

---

1. Patients, Admissions, Diagnoses, Prescriptions, Procedures, Lab Results, Diagnosis Code Dictionary, Procedure Code Dictionary, Lab Code Dictionary
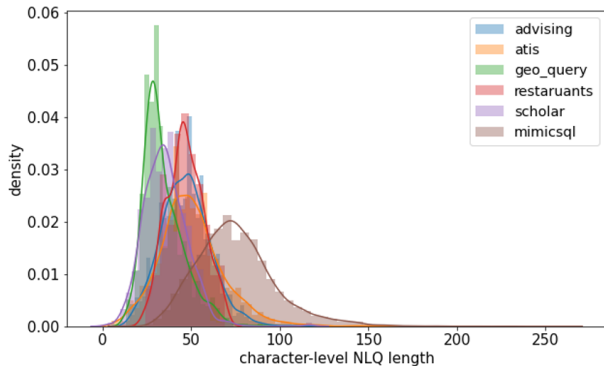
Figure 3: The distributions of the character-level NLQ length over various domain-specific NLQ2SQL datasets.

cause users to make typographical errors more often than the existing datasets. Therefore, in order to simulate such realistic scenarios, we created three MIMICSQL* variants, modified with different levels of NLQ typos: noise-*weak*, noise-*moderate*, and noise-*strong*. The details about the noise generation process are presented in the following section.

4.1.2. NOISE GENERATION

Following Kemighan et al. (1990), we adopt four different types of common typos: Reversal, Substitution, Deletion, and Insertion. When we corrupt a single word, we use one of the four types with the fixed ratio of 50%, 20%, 15%, and 15%, respectively. The detailed descriptions of four types are as follows:

- Reversal: Flip two adjacent characters in a word, such as "number" → "numbre".
- Substitution: Mistype a nearby key on a keyboard, such as "diagnosis" → "diagnowis".
- Deletion: Delete a random character of the word, such as "acetylcysteine" → "acetylcyseine".
- Insertion: Insert a character into the word, such as "coronary" → "copronary".

Based on the four pre-defined types of typos and their fixed ratios, we corrupt the target dataset with our noise generator via the following process. First, given a single natural language question, we split the sentence into individual words. For each word, the noise generator determines whether to make a typo based on its length and uniformly sampled probability p. If the generator determines to create a typo for that word, one of the four typo types are applied. The noise generator repeats this process for all NLQs in the target dataset.

We also make heuristic rules to prevent a sentence or its specific values from losing its original meaning due to typos. 1) The date-time value (e.g. "16:00:00") should not be changed; 2) The numerical value (e.g. "47" for age value) should not be changed; 3) The short word, smaller than the minimum length hyperparameter (l_{min}), should not be changed.

By using our noise generator, we apply three different degrees of noise (noise-*weak*, noise-*moderate*, and noise-*strong*) to MIMICSQL*, and use them only at the evaluation phase. Note that noise-*weak* contains approximately 5% of corrupted words in a sentence on average, noise-*moderate* about 10%, and noise-*strong* about 15%. The detailed algorithm is presented in the Appendix A.

We believe our three-level typo-ridden approach can reasonably represent realistic scenarios on MIMICSQL* for four reasons: 1) Cucerzan and Brill (2004) reports that users make 10-15% spelling errors in their queries when using search engines; 2) Following Hagiwara and Mita (2020), we confirm that real-world typo datasets have misspellings of approximately 10%; 3) MIMICSQL* has the longest natural language question compared to existing domain-specific NLQ2SQL datasets, due to the complex medical terms, as shown in Figure 3; 4) We found roughly a dozen instances with real-world typos in the MIMICSQL* *test* dataset, despite the fact the dataset went through manual inspection during construction. Also, we observed all their typos (*e.g.* "ethnicty", "nymber") occur at the character-level, confirming the realism of our synthetic noise injection method.

4.1.3. COMPARISON METHODS

We compare our model (**UniQA**) with the following baseline models. In all experiments, all models were trained with five random seeds, and we report the mean and the standard deviation. Further implementation and hyperparameter details are provided in the Appendix A.

Seq2Seq + Attention Seq2Seq with attention (Luong et al., 2015) consists of a bidirectional LSTM encoder and an LSTM decoder. Following the original paper, we apply the attention mechanism in this model. It should be noted this model cannot handle the out-of-vocabulary (OOV) tokens. We simply denote the model as Seq2Seq.

TREQS TREQS (Wang et al., 2020b) is the state-of-the-art NLQ2SQL model on MIMICSQL*. This LSTM-based encoder-to-decoder model uses temporal attention on NLQ, dynamic attention on SQL to

capture the condition values accurately, and copy-mechanism to resolve the OOV problem.

E-to-D TREQS is also an encoder-to-decoder model, but it does not use self-attention layers, and the model size is not comparable to ours. Accordingly, we adopt BERT2BERT (Rothe et al., 2020) to utilize the pre-trained BERT(6-layer, 768-hidden, 12-head) in both the encoder and the decoder, making the number of model parameters similar to UniQA. Its structure is exactly the same as the Transformer (Vaswani et al., 2017).

E-to-D + IM To provide a fair opportunity for the E-to-D model with the input masking strategy (IM), we trained the E-to-D model with masked language modeling on the encoder side while using the autoregressive modeling on the decoder side.

E-as-D In order to verify the effectiveness of the IM strategy, we use the vanilla encoder-as-decoder model, initialized with BERT(12-layer, 768-hidden, 12-head).

4.1.4. Evaluation Metrics

To evaluate the QA performance of different NLQ2Query models, we use the same evaluation metrics as described in previous works (Wang et al., 2020b; Park et al., 2021). 1) *Logical Form Accuracy* (Acc_{LF}) is computed by comparing the generated SQL/SPARQL queries with the true SQL/SPARQL queries token-by-token; 2) *Execution Accuracy* (Acc_{EX}) represents the matching ratio between results from executing the generated query and the results from executing the ground truth query. Note that it is possible for incorrect queries to luckily return correct results and affect this metric (*e.g.* when the target result is 0 or Null); 3) *Structural Accuracy* (Acc_{ST}) is equivalent to Acc_{LF} except that the condition value tokens (e.g. numeric values or string values) are ignored, therefore focusing on the SQL/SPARQL syntactic structure only.

4.1.5. Recovering for condition values

In addition to Input Masking, we use the condition value recovery technique used in Wang et al. (2020b) to better handle condition values often containing complex medical terms. After the NLQ2Query model generates a query, this technique is used to compare the condition values in the generated query to the existing values in the database. Then, the condition values in the generated query are replaced with the most similar (or identical) values in the database. For

Table 1: Test-set results on MIMICSQL*. We report the mean and standard deviation of the three evaluation metrics (Acc_{LF}, Acc_{EX}, and Acc_{ST}) over 5 random seeds. Note that E-to-D=Encoder-to-Decoder, E-as-D=Encoder-as-Decoder, IM=Input Masking.

Method	Test Performance for MIMICSQL*		
	Acc_{LF}	Acc_{EX}	Acc_{ST}
Before Recovering			
Seq2Seq	0.128 (0.077)	0.263 (0.066)	0.338 (0.030)
TREQS	0.604 (0.008)	0.694 (0.004)	0.799 (0.008)
E-to-D	0.832 (0.006)	0.884 (0.005)	0.900 (0.006)
E-to-D + IM	0.790 (0.006)	0.854 (0.002)	0.878 (0.007)
E-as-D	**0.856** (0.008)	**0.900** (0.007)	0.894 (0.003)
UniQA (E-as-D + IM)	0.849 (0.011)	0.895 (0.009)	**0.905** (0.013)
After Recovering			
Seq2Seq	0.136 (0.081)	0.242 (0.071)	0.338 (0.030)
TREQS	0.740 (0.006)	0.822 (0.009)	0.799 (0.008)
E-to-D	0.867 (0.006)	0.920 (0.008)	0.900 (0.006)
E-to-D + IM	0.792 (0.015)	0.875 (0.004)	0.878 (0.007)
E-as-D	0.878 (0.006)	0.928 (0.004)	0.894 (0.003)
UniQA (E-as-D + IM)	**0.882** (0.012)	**0.934** (0.008)	**0.905** (0.013)

example, if the user asks for the number of patients with *essential hypertension*, then the generated query will contain the condition value *essential hypertension*. But if this value does not exist in the database, the recovery technique will calculate the ROUGE-L scores between *essential hypertension* with all values in the database. Then the closest value, for example *essential hypertensive disorder*, is chosen and replaced with *essential hypertension*, thus making the generated query executable.

Note that both the recovery technique and the proposed IM training strategy have a similar purpose, namely handling complex medical terms. However, the recovery technique is a post-processing technique for the generated SQL/SPARQL query. If the generated query is an incorrect query to begin with due to noisy NLQ, the recovery technique would only rectify the condition values, but the entire query would still be incorrect. From this perspective, another role of IM, which not only adjusts typos but also captures query structure, can be combined with this recovery technique, where both techniques can complement each other effectively. We will further discuss this in the following experiment and analysis section.

4.2. Experiments Results

4.2.1. Results from MIMICSQL*

Model performance on MIMICSQL* test set are shown in Table 1. After applying the recovering technique for the condition value, we can observe

Table 2: Test-set results on NOISY MIMICSQL*. We report the mean and standard deviation of the three evaluation metrics (Acc_{LF}, Acc_{EX}, and Acc_{ST}) over 5 random seeds. Based on the probability of corrupting words on average in a sentence, we refer to 5% as noise-*weak*, 10% as noise-*moderate*, and 15% as noise-*strong*.

Method	noise-*weak* (5% typo prob.)			noise-*moderate* (10% typo prob.)			nosie-*strong* (15% typo prob.)		
	Acc_{LF}	Acc_{EX}	Acc_{ST}	Acc_{LF}	Acc_{EX}	Acc_{ST}	Acc_{LF}	Acc_{EX}	Acc_{ST}
Before Recovering									
Seq2Seq	0.096 (0.062)	0.226 (0.056)	0.264 (0.032)	0.063 (0.033)	0.169 (0.033)	0.173 (0.017)	0.044 (0.024)	0.135 (0.023)	0.119 (0.020)
TREQS	0.418 (0.011)	0.546 (0.010)	0.665 (0.011)	0.281 (0.019)	0.412 (0.024)	0.559 (0.031)	0.228 (0.020)	0.363 (0.024)	0.483 (0.041)
E-to-D	0.709 (0.007)	0.783 (0.006)	0.839 (0.009)	0.554 (0.011)	0.648 (0.013)	0.703 (0.008)	0.456 (0.015)	0.551 (0.016)	0.639 (0.013)
E-to-D + IM	0.665 (0.027)	0.754 (0.025)	0.791 (0.025)	0.490 (0.031)	0.593 (0.027)	0.638 (0.036)	0.417 (0.032)	0.522 (0.031)	0.576 (0.032)
E-as-D	0.654 (0.005)	0.738 (0.005)	0.850 (0.004)	0.510 (0.016)	0.615 (0.017)	0.749 (0.016)	0.382 (0.018)	0.493 (0.016)	0.684 (0.025)
UniQA (E-as-D + IM)	**0.740** (0.027)	**0.807** (0.021)	**0.877** (0.016)	**0.610** (0.035)	**0.691** (0.028)	**0.784** (0.028)	**0.504** (0.032)	**0.602** (0.028)	**0.731** (0.024)
After Recovering									
Seq2Seq	0.103 (0.065)	0.201 (0.059)	0.264 (0.032)	0.066 (0.035)	0.141 (0.036)	0.173 (0.017)	0.046 (0.027)	0.105 (0.026)	0.119 (0.020)
TREQS	0.545 (0.012)	0.668 (0.009)	0.665 (0.011)	0.397 (0.025)	0.531 (0.029)	0.559 (0.031)	0.325 (0.032)	0.464 (0.033)	0.483 (0.041)
E-to-D	0.783 (0.011)	0.848 (0.010)	0.839 (0.009)	0.628 (0.012)	0.714 (0.015)	0.703 (0.008)	0.533 (0.013)	0.629 (0.012)	0.639 (0.013)
E-to-D + IM	0.681 (0.028)	0.773 (0.021)	0.791 (0.025)	0.505 (0.030)	0.610 (0.027)	0.638 (0.036)	0.433 (0.032)	0.546 (0.033)	0.576 (0.032)
E-as-D	0.818 (0.006)	0.880 (0.004)	0.850 (0.004)	0.692 (0.014)	0.767 (0.015)	0.749 (0.016)	0.602 (0.021)	0.672 (0.020)	0.668 (0.050)
UniQA (E-as-D + IM)	**0.825** (0.018)	**0.895** (0.013)	**0.877** (0.016)	**0.695** (0.028)	**0.775** (0.023)	**0.784** (0.028)	**0.612** (0.021)	**0.709** (0.018)	**0.731** (0.024)

Table 3: Test-results on MIMICSPARQL*. We report the mean and standard deviation of the three evaluation metrics (Acc_{LF}, Acc_{EX}, and Acc_{ST}) over 5 random seeds.

Method	Test Performance for MIMICSPARQL*		
	Acc_{LF}	Acc_{EX}	Acc_{ST}
Before Recovering			
Seq2Seq	0.219 (0.053)	0.336 (0.042)	0.339 (0.010)
TREQS	0.603 (0.020)	0.702 (0.014)	0.761 (0.021)
E-to-D	0.823 (0.006)	0.875 (0.004)	0.890 (0.003)
E-to-D + IM	0.776 (0.008)	0.842 (0.011)	0.864 (0.006)
E-as-D	**0.866** (0.005)	**0.909** (0.004)	0.903 (0.002)
UniQA (E-as-D + IM)	0.853 (0.009)	0.897 (0.01)	**0.909** (0.011)
After Recovering			
Seq2Seq	0.222 (0.055)	0.346 (0.044)	0.339 (0.010)
TREQS	0.610 (0.022)	0.715 (0.012)	0.761 (0.021)
E-to-D	0.864 (0.005)	0.912 (0.004)	0.890 (0.003)
E-to-D + IM	0.792 (0.007)	0.860 (0.008)	0.864 (0.006)
E-as-D	**0.896** (0.003)	**0.937** (0.006)	0.903 (0.002)
UniQA (E-as-D + IM)	0.891 (0.008)	0.935 (0.005)	**0.909** (0.011)

Table 4: Test-results on NOISY MIMICSPARQL* with the degree of noise-*moderate*. We report the mean and standard deviation of the three evaluation metrics (Acc_{LF}, Acc_{EX}, and Acc_{ST}) over 5 random seeds.

Method	noise-*moderate* (10% typo prob.)		
	Acc_{LF}	Acc_{EX}	Acc_{ST}
Before Recovering			
Seq2Seq	0.093 (0.016)	0.178 (0.019)	0.178 (0.019)
TREQS	0.277 (0.016)	0.426 (0.009)	0.494 (0.019)
E-to-D	0.547 (0.009)	0.639 (0.010)	0.704 (0.016)
E-to-D + IM	0.534 (0.025)	0.631 (0.031)	0.678 (0.032)
E-as-D	0.504 (0.006)	0.609 (0.010)	0.715 (0.024)
UniQA (E-as-D + IM)	**0.646** (0.015)	**0.725** (0.015)	**0.798** (0.016)
After Recovering			
Seq2Seq	0.094 (0.017)	0.182 (0.020)	0.178 (0.019)
TREQS	0.283 (0.017)	0.434 (0.008)	0.494 (0.019)
E-to-D	0.618 (0.017)	0.708 (0.021)	0.704 (0.016)
E-to-D + IM	0.546 (0.024)	0.644 (0.029)	0.678 (0.032)
E-as-D	0.668 (0.018)	0.740 (0.022)	0.715 (0.024)
UniQA (E-as-D + IM)	**0.725** (0.010)	**0.799** (0.011)	**0.798** (0.016)

that UniQA outperforms the previous state-of-the-art model TREQS by a significant margin 14.2%. Thanks to IM, by simply applying the input masking to the vanilla E-as-D model, we can observe that our model has the best structural accuracy over all baselines. We also confirm that the performance gain is higher than E-as-D in logical form accuracy compared to before applying the recovering technique, indicating that the recovering technique (*i.e.* replace with proper condition value after the decoding stage) and IM's ability (*i.e.* address typos preemptively at the decoding stage, and capture the SQL syntax) worked in harmony effectively.

4.2.2. Results from Noisy MIMICSQL*

Model performance for noisy MIMICSQL* are shown in Table 2. As the probability of typos rises from weak to strong, we observe that the performances of all models decrease across three evaluation metrics. Along with our input masking strategy, we can observe UniQA outperforms all baseline models, especially outperforming the previous state-of-the-art model TREQS by a significant margin 25.8%. Furthermore, UniQA always recorded the best structural accuracy regardless of the degree of typos. This means that the additional post-processing by the end-user (*e.g.* manually typing in correct condition values) has the potential for further increasing the model performance, since structural accuracy is the upper bound of logical form accuracy.

5. Discussion

5.1. Graph-based EHR Question Answering

In terms of viewing EHR as a massive knowledge graph (KG) rather than multiple relational tables, there is another publicly available EHR-QA dataset, MIMICSPARQL*. Although MIMICSPARQL* has the different style of queries (*i.e.* SPARQL), the answer is eventually the same as MIMICSQL* because it has the same NLQs as MIMICSQL* and tables are transformed into a knowledge graph without any information loss. To evaluate models over graph-based EHR-QA, we conducted the same experiments as MIMICSQL*, with the original MIMICSPARQL* and its noisy variants.

As shown in Table 3 and Table 4, we can observe UniQA shows the consistent results on MIMIC-SPARQL* and its variants. As demonstrated in all previous experiments, UniQA has the highest structural accuracy over all baselines. In addition, Park et al. (2021) demonstrate that the graph-based approach is more suitable for EHR-QA. We also empirically demonstrate that the overall performance of MIMICSPARQL* is higher than MIMICSQL*, consistent with the empirical results from Park et al. (2021).

5.2. Why IM does not help in E-to-D case

As shown in Table 2, the performance of the E-to-D model decreases when the model is combined with the input masking strategy. We believe this originates from the architectural difference between E-to-D and E-as-D. In contrast to E-as-D where a single transformer model acts as both encoder and decoder, E-to-D employs two distinct encoder and decoder. Therefore when E-as-D is combined with IM, both encoder and decoder are trained with the reconstruction loss. E-to-D, however, when combined with IM, encoder and decoder are trained with two distinct losses (*i.e.* encoder with reconstruction loss, decoder with autoregressive loss), which seems to yield a negative effect rather than a synergistic effect.

5.3. Qualitative Comparison between Generated Queries

We demonstrate the qualitative results to study differences between models and how each model generates the SQL query given the noisy input question. As shown in Figure 4, we present the ground truth and generated queries by six models given two NLQs

with a typo. All results are generated during the evaluation phase on the NOISY MIMICSQL* dataset with the degree of noise-*moderate*.

On the left side, the word *"ferrous gluconate"* in the NLQ, used as the condition value for the SQL query, is corrupted by a single deletion. Due to this typo, all models except UniQA have errors in the condition value part. Interestingly, we can see that E-to-D generates condition value that is not present in the noisy NLQ, but E-as-D copies the incorrect condition value including a typo. Thanks to the IM strategy, unlike the vanilla E-as-D model, UniQA correctly adjusts the corrupted word to its original condition value.

On the right side, the word *"language"* in the NLQ, used as the target column in the SQL query, is also corrupted by a single deletion. This simple deletion makes all models except UniQA generate incorrect SQL queries with incorrect table names or column names. Here we can confirm that UniQA can capture table and column names well even when given the noisy input. Based on the two cases discussed above, it can be seen that UniQA is robust to the typos regardless of its positions in the input question, namely be it medical terms or other lengthy words.

5.4. Error Analysis

To gain intuition and understand the challenging points in EHR-QA, we conducted an error analysis on failure cases (*i.e.* 151 samples) made by UniQA for MIMICSQL* without any noise.

Insufficient information in the question This most popular failure type occurs because the given natural language question provides insufficient or implicit information to generate accurate queries. For example, in the question *how many of the male patients had icd9 code 8842?*, the *icd9 code* may refer the *diagnoses_icd9_code* or the *procedure_icd9_code* column, so the model might generate an incorrect query. Another example is incorrectly decoding non-specific questions such as *specify details of icd9 code 4591*, which should be decoded to retrieve both short and long names of code 4591, but the model would retrieve only the long name. If the model can interact with the user (*e.g.* ask clarification questions to the user), this failure type can be significantly alleviated.

Handling paraphrased questions Questions can be semantically similar to the training samples, but lexically very dissimilar. In this case the model can

noisy NLQ	how many patients were given the drug **ferros** gluconate? (typo: ferrous → ferros)	which **laguage** does cynthia gomez understand? (typo: language → laguage)
Ground Truth	select count (distinct patients.subject_id) 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 = "ferrous gluconate"	select admissions.language from admissions inner join patients on admissions.subject_id = patients.subject_id where patients.name = "cynthia gomez"
Seq2Seq	select count (distinct patients.subject_id) 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 = <UNK> <UNK>	select d_icd_diagnoses.short_title from d_icd_diagnoses inner join diagnoses on d_icd_diagnoses.icd9_code = diagnoses.icd9_code where patients.name = <UNK> <UNK>
TREQS	select count (distinct patients.subject_id) from patients inner join admissions on patients.subject_id= admissions.subject_id inner join prescriptions on admissions.hadm_id=prescriptions.hadm_id where admissions.age < "44" and prescriptions.drug = "glucohate glucohate"	select count (distinct patients.subject_id) 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 = "gomez sibject"
E-to-D	select count (distinct patients.subject_id) 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 = "ferrous sulfate"	select admissions.age from admissions inner join patients on admissions.subject_id = patients.subject_id where patients.name = "cynthia gomez"
E-to-D+IM	select count (distinct patients.subject_id) 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 = "ferrous sulfate"	select admissions.insurance from admissions inner join patients on admissions.subject_id = patients.subject_id where patients.name = "cynthia gomez"
E-as-D	select count (distinct patients.subject_id) 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 = "ferros gluconate"(†)	select admissions.marital_status from admissions inner join patients on admissions.subject_id = patients.subject_id where patients.name = "cynthia gomez"
UniQA (E-as-D + IM)	select count (distinct patients.subject_id) 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 = "ferrous gluconate"	select admissions.language from admissions inner join patients on admissions.subject_id = patients.subject_id where patients.name = "cynthia gomez"

Figure 4: SQL Queries generated by different models given the same noisy NLQ. The typos in the NLQs are marked in bold, and incorrectly predicted tokens in the generated SQL queries are highlighted in red. The correctly predicted SQL tokens, associated with the typo in the NLQ, are highlighted in blue. (†) denotes a sample that can be corrected by the recovering technique. Note that the recovering technique can help only one case out of 10 incorrectly generated queries, indicating the effectiveness of the input masking strategy for noisy NLQs.

have a hard time generating correct queries. For example, given the question *find the number of patients who are no longer alive*, the model must generate the condition *patients.expire_flag=1*. This would be considerably more difficult since the model was trained with samples such as *find the number of patients who expired*. This failure type can be potentially alleviated by using a very powerful pre-trained language model.

Rare question types The model has a hard time handling question types that rarely occur in the training set. For example, multi-part questions such as *When was patient id XXX admitted? Specify time and location* occur only twice in the training set, and this provides very little chance for the model to learn to correctly answer this question type.

6. Conclusion

In this work, we proposed UniQA, a unified Encoder-as-Decoder model with the input masking technique to cope with EHR-QA containing complex medical terminology. We applied UniQA on a large publicly available NLQ2Query dataset, MIMICSQL* and demonstrated significantly superior performance over the previous state-of-the-art method. In addition, given the same experimental settings, our model showed consistent superior results for the graph-based EHR-QA dataset, MIMICSPARQL*. We plan to extend our model to incorporate user interaction as discussed in the error analysis and further address more domain-specific challenges such as abbreviated terms in the future.

References

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. Ryansql: Recursively

applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. *arXiv preprint arXiv:2004.03125*, 2020.

Silviu Cucerzan and Eric Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 293–300, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-3238.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, 2018.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 13063–13075, 2019.

Alessandra Giordani and Alessandro Moschitti. Translating questions to SQL queries with generative parsers discriminatively reranked. In *Proceedings of COLING 2012: Posters*, pages 401–410, Mumbai, India, December 2012. The COLING 2012 Organizing Committee. URL https://www.aclweb.org/anthology/C12-2040.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, 2019.

Masato Hagiwara and Masato Mita. GitHub typo corpus: A large-scale multilingual dataset of mis-spellings and grammatical errors. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 6761–6768, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.835.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*, 2019.

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*, 3(1):1–9, 2016.

Mark D Kemighan, Kenneth Church, and William A Gale. A spelling correction program based on a noisy channel model. In *COLING 1990 Volume 2: Papers presented to the 13th International Conference on Computational Linguistics*, 1990.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Fei Li and HV Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

Anusri Pampari, Preethi Raghavan, Jennifer Liang, and Jian Peng. emrqa: A large corpus for question answering on electronic medical records. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2357–2368, 2018.

Junwoo Park, Youngwoo Cho, Haneol Lee, Jaegul Choo, and Edward Choi. Knowledge graph-based question answering with electronic health records. In *Machine Learning for Healthcare Conference*, pages 36–53. PMLR, 2021.

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

Patti Price. Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*, 1990.

Chris Quirk, Raymond Mooney, and Michel Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 878–888, 2015.

Preethi Raghavan, Jennifer J Liang, Diwakar Mahajan, Rachita Chandra, and Peter Szolovits. emrkbqa: A clinical knowledge-base question answering dataset. In *Proceedings of the 20th Workshop on Biomedical Language Processing*, pages 64–73, 2021.

Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. Leveraging pre-trained checkpoints for sequence generation tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280, 2020.

Simon Šuster and Walter Daelemans. Clicr: A dataset of clinical case reports for machine reading comprehension. In *Proceedings of NAACL-HLT*, pages 1551–1563, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Ratsql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, 2020a.

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

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, 2018.

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

Ming Zhu, Aman Ahuja, Da-Cheng Juan, Wei Wei, and Chandan K Reddy. Question answering with long multiple-span answers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 3840–3849, 2020.

Appendix A. Implementation Details

A.1. Noise generator

Algorithm 1: Noise Generator

Input: a question $Q = \{q_1, q_2, \ldots, q_n\}$, noise
      rate r_{noise}, min word length l_{min}
Output: a noisy question $\tilde{Q} = \{\tilde{q_1}, \tilde{q_2}, \ldots, \tilde{q_n}\}$
$\tilde{Q} \leftarrow [\,]$
for $i \leftarrow 1$ **to** n **do**
    $p \leftarrow \mathrm{Uniform}(0, 1)$
    if $p \cdot \log[\mathrm{LEN}(q_i)] \leq r_{noise}$ **then**
        $r \leftarrow \mathrm{Uniform}(0, 1)$
        if ($\mathrm{ISNUMERIC}(q_i)$ ***or*** $\mathrm{ISDATE}(q_i)$ ***or***
        $\mathrm{LEN}(q_i) \leq l_{min}$) **then**
          $\tilde{q_i} \leftarrow q_i$
        else if $0 \leq r < 0.15$ **then**
          $\tilde{q_i} \leftarrow \mathrm{EXTRALETTER}(q_i)$
        else if $0.15 \leq r < 0.30$ **then**
          $\tilde{q_i} \leftarrow \mathrm{MISSINGLETTER}(q_i)$
        else if $0.30 \leq r < 0.50$ **then**
          $\tilde{q_i} \leftarrow \mathrm{WRONGLETTER}(q_i)$
        else
          $\tilde{q_i} \leftarrow \mathrm{REVERSEDLETTER}(q_i)$
        end
    else
        $\tilde{q_i} \leftarrow q_i$
    end
    $\tilde{Q}.\mathrm{ADD}(\tilde{q_i})$
end

A.2. Implementation details

We implement our model and baseline models with PyTorch Lightning [2] and HuggingFace's transformers[3]. In the case of TREQS, we utilized the official code[4] written by the origin authors. We use the original BERT as our pre-trained model. For the fair comparison, we adjust the number of self-attention layer of encoder and decoder in Encoder-to-Decoder model to half of ours. We use Adam optimizer (Kingma and Ba, 2014) with the learning rate set to 3×10^{-5} and batch size set to 32.

A.3. Hyperparameters

In order to make an accurate comparison with the baseline models, Seq2seq model and TREQS model were imported from Park et al. (2021), and hyperparameters were also imported with the same value.

We trained our models on two types of GPU environments: NVIDIA Tesla T4 and NVIDIA GeForce RTX-3090. Also, torch version is 1.7.0, and CUDA version is 11.1. Other hyparparameters are presented in Table 5.

Appendix B. Results

B.1. Quantitative results of noisy MIMICSPARQL*

We provide the quantitative results on MIMIC-SPARQL* dataset with the different degree of noise in the Table 6.

B.2. Qualitative results of MIMICSQL*

We provide the qualitative results on MIMICSQL* dataset in the Figure 5.

---

2. https://www.pytorchlightning.ai
3. https://huggingface.co/transformers/
4. https://github.com/wangpinggl/TREQS

Hyperparameters	Seq2Seq	TREQS	E-to-D (+IM)	E-as-D (+IM)
Hidden dimension	256	256(enc) + 256(dec)	768	768
Learning rate	5×10^{-4}	5×10^{-4}	3×10^{-5}	3×10^{-5}
LR Scheduler	StepLR(step size = 2, step decay = 0.8)	StepLR(step size = 2, step decay = 0.8)	Linear decay	Linear decay
Batch size	16	64	32	32
Epochs	20	20	100 (w/ early stop)	100 (w/ early stop)
Seed	1, 12, 123, 1234, 42	1, 12, 123, 1234, 42	1, 12, 123, 1234, 42	1, 12, 123, 1234, 42
Beam size	-	5	5	5
Seq2seq-LM prob.	-	-	-	0.3
Input Masking prob.	-	-	0.2 (only for IM)	0.2 (only for IM)

Table 5: Hyperparameters for training several models.

Method	Testing-weak (5% typo prob.)			Testing-moderate (10% typo prob.)			Testing-strong (15% typo prob.)		
	Acc_{LF}	Acc_{EX}	Acc_{ST}	Acc_{LF}	Acc_{EX}	Acc_{ST}	Acc_{LF}	Acc_{EX}	Acc_{ST}
Before Recovering									
Seq2Seq	0.159 (0.039)	0.265 (0.029)	0.260 (0.011)	0.093 (0.016)	0.178 (0.019)	0.178 (0.019)	0.070 (0.016)	0.140 (0.018)	0.127 (0.014)
TREQS	0.419 (0.020)	0.551 (0.015)	0.630 (0.018)	0.277 (0.016)	0.426 (0.009)	0.494 (0.019)	0.215 (0.009)	0.376 (0.011)	0.422 (0.015)
E-to-D	0.704 (0.011)	0.778 (0.010)	0.833 (0.016)	0.547 (0.009)	0.639 (0.010)	0.704 (0.016)	0.449 (0.020)	0.549 (0.019)	0.643 (0.029)
E-to-D + IM	0.679 (0.015)	0.764 (0.019)	0.801 (0.015)	0.534 (0.025)	0.631 (0.031)	0.678 (0.032)	0.465 (0.012)	0.565 (0.009)	0.621 (0.020)
E-as-D	0.660 (0.006)	0.741 (0.005)	0.837 (0.014)	0.504 (0.006)	0.609 (0.010)	0.715 (0.024)	0.385 (0.008)	0.493 (0.014)	0.648 (0.030)
UniQA (E-as-D + IM)	**0.754** (0.022)	**0.820** (0.018)	**0.877** (0.013)	**0.646** (0.015)	**0.725** (0.015)	**0.798** (0.016)	**0.535** (0.022)	**0.627** (0.023)	**0.726** (0.014)
After Recovering									
Seq2Seq	0.160 (0.040)	0.273 (0.030)	0.260 (0.011)	0.094 (0.017)	0.182 (0.020)	0.178 (0.019)	0.071 (0.016)	0.143 (0.017)	0.127 (0.014)
TREQS	0.426 (0.021)	0.562 (0.017)	0.630 (0.018)	0.283 (0.017)	0.434 (0.008)	0.494 (0.019)	0.221 (0.009)	0.382 (0.009)	0.422 (0.015)
E-to-D	0.777 (0.015)	0.847 (0.013)	0.833 (0.013)	0.618 (0.017)	0.708 (0.021)	0.704 (0.016)	0.526 (0.028)	0.620 (0.028)	0.643 (0.029)
E-to-D + IM	0.696 (0.012)	0.786 (0.016)	0.801 (0.015)	0.546 (0.024)	0.644 (0.029)	0.678 (0.032)	0.481 (0.011)	0.582 (0.010)	0.621 (0.020)
E-as-D	0.819 (0.012)	0.874 (0.016)	0.837 (0.014)	0.668 (0.018)	0.740 (0.022)	0.715 (0.024)	0.571 (0.022)	0.637 (0.031)	0.648 (0.030)
UniQA (E-as-D + IM)	**0.835** (0.012)	**0.896** (0.012)	**0.877** (0.013)	**0.725** (0.010)	**0.799** (0.011)	**0.798** (0.016)	**0.625** (0.009)	**0.706** (0.007)	**0.726** (0.014)

Table 6: QA Performance on NOISY MIMICSPARQL* natural questions with evaluated with logic form accuracy (Acc_{LF}), execution accuracy (Acc_{EX}), and the structural accuracy (Acc_{ST}). Based on the probability of noise, we refer to 5% as weak, 10% as moderate, and 15% as strong.

NLQ	how many patients with elective admission type had the procedure titled **other operations on heart and pericardium**?
Ground Truth	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 admissions.admission_type = "elective" and d_icd_procedures.long_title = **"other operations on heart and pericardium"**
Seq2Seq	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 admissions.admission_type = "elective" and d_icd_procedures.short_title = **"other <UNK> nec"**
TREQS	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 admissions.admission_type = "elective" and d_icd_procedures.long_title = **"other operations on on heart and pericardium"**
E-to-D	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 admissions.admission_type = "elective" and d_icd_procedures.long_title = **"other conversion of heart and pericardium"**
E-to-D + IM	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 admissions.admission_type = "elective" and d_icd_procedures.long_title = **"other diagnostic procedures on skin and subcutaneous tissue"**
E-as-D	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 admissions.admission_type = "elective" and d_icd_procedures.long_title = **"other operations on heart and pericardium"**
UniQA (E-as-D + IM)	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 admissions.admission_type = "elective" and d_icd_procedures.long_title = **"other operations on heart and pericardium"**

Figure 5: SQL Queries generated by different models given the same NLQ in MIMICSQL* test set. Conditional values corresponding to the medical term "*other operations on heart and pericardium*" mentioned in the NLQ are marked in bold. The incorrectly predicted tokens are highlighted in red color.