Text-to-SQL Generation for Question Answering on Electronic Medical Records

Ping Wang
Dept. of Computer Science
Virginia Tech, Arlington, VA
ping@vt.edu

Tian Shi
Dept. of Computer Science
Virginia Tech, Arlington, VA
tshi@vt.edu

Chandan K. Reddy
Dept. of Computer Science
Virginia Tech, Arlington, VA
reddy@cs.vt.edu

ABSTRACT

Electronic medical records (EMR) contain comprehensive patient information and are typically stored in a relational database with multiple tables. Effective and efficient patient information retrieval from EMR data is a challenging task for medical experts. *Question-to-SQL generation* methods tackle this problem by first predicting the SQL query for a given question about a database, and then, executing the query on the database. However, most of the existing approaches have not been adapted to the healthcare domain due to a lack of healthcare Question-to-SQL dataset for learning models specific to this domain. In addition, wide use of the abbreviation of terminologies and possible typos in questions introduce additional challenges for accurately generating the corresponding SQL queries. In this paper, we tackle these challenges by developing a deep learning based **TR**anslate-**E**dit Model for **Q**uestion-to-**S**QL (TREQS) generation, which adapts the widely used sequence-to-sequence model to directly generate the SQL query for a given question, and further performs the required edits using an attentive-copying mechanism and task-specific look-up tables. Based on the widely used publicly available electronic medical database, we create a new large-scale Question-SQL pair dataset, named *MIMICSQL*, in order to perform the Question-to-SQL generation task in healthcare domain. An extensive set of experiments are conducted to evaluate the performance of our proposed model on MIMICSQL. Both quantitative and qualitative experimental results indicate the flexibility and efficiency of our proposed method in predicting condition values and its robustness to random questions with abbreviations and typos.

CCS CONCEPTS

• **Information systems** → **Question answering**; • **Computing methodologies** → **Information extraction**; **Neural networks**; • **Applied computing** → **Health informatics**.

KEYWORDS

Sequence-to-sequence model, attention mechanism, pointer-generator network, electronic medical records, SQL query.

ACM Reference Format:
Ping Wang, Tian Shi, and Chandan K. Reddy. 2020. Text-to-SQL Generation for Question Answering on Electronic Medical Records. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3366423.3380120

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.
WWW '20, April 20–24, 2020, Taipei, Taiwan
© 2020 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC-BY 4.0 License.
ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380120

1 INTRODUCTION

Due to recent advances of data collection and storing techniques, a large amount of healthcare related data, typically in the form of electronic medical records (EMR), are accumulated everyday in clinics and hospitals. EMR data contain a comprehensive set of longitudinal information about patients and are usually stored in structured databases with multiple relational tables, such as demographics, diagnosis, procedures, prescriptions, and laboratory tests. One important mechanism of assisting doctors' clinical decision making is to directly retrieve patient information from EMR data, including patient-specific information (e.g., individual demographic and diagnosis information) and cohort-based statistics (e.g., mortality rate and prevalence rate). Typically, doctors interact with EMR data using searching and filtering functions available in rule-based systems that first turn any predefined-rule (front-end) to a SQL query (back-end), and then, return an answer. These systems are complicated and require special training before being used. They are also difficult to manage and extend. For example, the front-end needs to be adapted for newer functionalities. Therefore, doctors who depend on these systems cannot fully and freely explore EMR data. Another challenge for these systems is that the users have to first transform their questions to a combination of rules in the front-end, which is not convenient and efficient. For instance, if a doctor wants to know the number of patients who are under the age of 40 and suffering from diabetes, then, he/she may have to create two filters, one for disease and the other for age. An alternate way to solve this problem is to build a model that can translate this question directly to its SQL query, so that the doctor only needs to type his/her question as: "Give me the number of patients whose age is below 40 and have diabetes", in the search box to get the answer. Motivated by this intuition, we propose a new deep learning based model that can translate textual questions on EMR data to SQL queries (*Text-to-SQL generation*) without any assistance from a database expert. As a result, these systems can assist doctors with their clinical decisions more efficiently. Since the textual input in our task is a clinical question, we will refer to Text-to-SQL generation as *Question-to-SQL generation* from now onwards.

Recently, Question-to-SQL generation task has gained significant attention and found applications in a variety of domains, including WikiSQL [5, 36, 46] for Wikipedia, ATIS [12] about flight-booking, GeoQuery [6] about US geography, and Spider [44] for general purpose cross-domain applications. There are also few works in this line of research in healthcare domain [2, 26]. Broadly speaking, various approaches for Question-to-SQL generation task belong

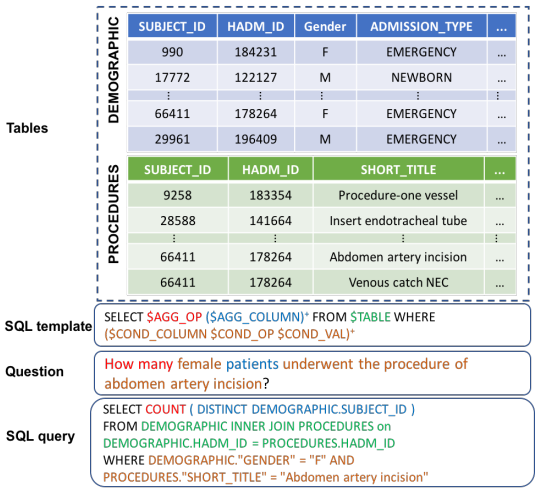


Figure 1: An example from MIMICSQL. The two tables, namely, Demographics and Diagnoses, are used to answer the question. Different colors are used to show the correspondence between various components in source question, targeted SQL query, and SQL template.

to one of the following two categories: (1) *Semantic parsing or slot-filling methods* [5, 9, 36, 38, 42, 43]: These models make use of semantic and syntactic information in a question and a table schema to generate a SQL logic form (parsing tree), which can be easily converted to the corresponding executable query. However, they strongly depend on pre-defined templates, which limits their applications in generating complex SQL queries. (2) *Language generation methods* [35, 45, 46]: These models leverage the power of language generation models and can directly generate SQL queries without building pre-defined SQL templates [20]. Therefore, they can be easily applied to produce complex SQL queries, regardless of the number of tables and columns involved. However, the predicted SQL queries may not be executable due to limited and inaccurate information from questions (e.g., a random question with typos and missing keywords). Moreover, it is difficult to interpret language generation models and their outputs.

Many recent Question-to-SQL models have been primarily benchmarked on WikiSQL [5, 36, 46] and Spider [3, 9, 43] datasets. In WikiSQL, questions are asked against databases with a single table, and every SQL query is composed of a "SELECT" (column/aggregation) clause along with a "WHERE" (condition) clause that consists of one or multiple conditions. Different from WikiSQL, the Spider dataset contains lots of complex and nested queries (e.g., "GROUP BY" and "HAVING") which may involve multiple tables [44]. Some recent studies have shown that several models which perform well on WikiSQL achieve poor results on Spider [9, 45]. It indicates that models for Question-to-SQL generation on single-table databases cannot be simply adapted to database with multiple relational tables. For Spider dataset, the current Question-to-SQL generation task focuses on generating SQL queries without actual values for "WHERE" conditions, which means models are only required to predict SQL structures and parse corresponding table and column names. However, even if a model can produce high quality SQL structures and columns, condition value generation may still be the bottleneck in producing correct and executable SQL queries [38].

Another issue with WikiSQL and Spider dataset is that most words (78% for WikiSQL and 65% for Spider) in database schema in development/testing sets have appeared in the training set [9]. Therefore, it is not feasible to apply the models trained on the Spider dataset to some other domains like chemistry, biology, and healthcare. Specific to healthcare domain, Question-to-SQL generation for EMR data is still under-explored. There are three primary challenges: (1) **Medical terminology abbreviations.** Due to the wide use of abbreviation of medical terminology (sometimes typos), it is difficult to match keywords in questions to those in database schema and table content. (2) **Condition value parsing and recovery.** It is still a challenging task to extract condition values from questions and recover them based on table content, especially in the appearance of medical abbreviations. (3) **Lack of large-scale healthcare Question-to-SQL dataset.** Currently, there is no dataset available for the Question-to-SQL task in the healthcare domain.

To tackle these challenges, we first generated a large-scale healthcare Question-to-SQL dataset, namely MIMICSQL, that consists of 10,000 Question-SQL pairs, by using the publicly available real-world Medical Information Mart for Intensive Care III (MIMIC III) dataset [8, 13] and leveraging the power of *crowd-sourcing*. An illustrative example in MIMICSQL is provided in Figure 1 to illustrate various components of the dataset. Based on MIMICSQL data, we further propose a language generation based Translate-Edit model, which can first translate a question to the corresponding SQL query, and then, retrieve condition values based on the question and table content. The editing meta-algorithms make our model more robust to randomly asked questions with insufficient information and typos, and make it practical to retrieve and recover condition values effectively. The major contributions of this paper are as follows:

- Propose a two-stage **TR**anslate-**E**dit Model for **Q**uestion-to-**S**QL (TREQS) generation model, which consists of three main components: (1) Translating an input question to a SQL query using a Seq2Seq based model, (2) Editing the generated query with attentive-copying mechanism, and (3) Further editing it with task-specific look-up tables.
- Create a large-scale dataset for Question-to-SQL task in healthcare domain. MIMICSQL has two subsets, in which the first set is composed of template questions (machine generated), while the second consists of natural language questions (human annotated). To the best of our knowledge, it is the first dataset for healthcare question answering on EMR data with multi-relational tables.
- Conduct an extensive set of experiments on MIMICSQL dataset for both template questions and natural language questions to demonstrate the effectiveness of the proposed model. Both qualitative and quantitative results indicate that it outperforms several baseline methods.

The rest of this paper is organized as follows. Section 2 describes some prior work related to Question-to-SQL generation, and differentiate our work from other existing works. Section 3 provides a comprehensive description of the MIMICSQL data generation process. Section 4 provides the details of the proposed translate-edit model. Section 5 shows the comparison of our proposed model with the state-of-the-art methods by analyzing both quantitative and qualitative results. Finally, we conclude the paper in Section 6.

2 RELATED WORK

Question-to-SQL generation is a sub-task of semantic parsing, which aims at translating a natural language text to a corresponding formal semantic representation, including SQL queries, logic forms and code generation [4, 37]. It has attracted significant attention in various applications, including WikiSQL [36, 46] for Wikipedia, ATIS [12] about flight-booking, GeoQuery [6] about US geography and Spider [44] about cross-domain. In the literature of Question-to-SQL generation, a common way is to utilize a SQL structure-based sketch with multiple slots and formulate the problem as a slot filling task [5, 29, 36, 42, 46] by incorporating some form of pointing/copying mechanism [33]. Seq2SQL method [46] is an augmented pointer network based framework and mainly prunes the output space of the target query by leveraging the unique structures of SQL commands. SQLNet method [36] is proposed to avoid the "order-matter" problem in the condition part by using a sketch-based approach instead of the sequence-to-sequence (Seq2Seq) based method. By further improving SQLNet, TYPESQL method [42] captures the rare entities and numbers in natural language questions by utilizing the type information. The two-stage semantic parsing method named Coarse2Fine [5] first generates a sketch of a given question and then fills in missing details based on both input question and the sketch. Recently, several semantic parsing methods [3, 9, 43] are also proposed on Spider to tackle the problem across different domains. One limitation of these methods is that they are highly dependent on the SQL structure and the lexicons, and thus cannot efficiently retrieve the condition values. Therefore, compared to other components, the performance of most semantic parsing methods in predicting condition values tend to be relatively low and these methods primarily focus on predicting correct SQL structures and columns, especially for the cross-domain problem present in the recently released Spider data [44].

To overcome the disadvantage of slot filling methods, Seq2Seq based methods [4, 20, 32, 34] are proposed to tackle this challenge by directly generating the targeted SQL queries. More specifically, Seq2Seq based methods first encode input questions into vector representations and then decode the corresponding SQL conditioned on the encoded vectors. A type system of SQL expressions is applied in the deep Seq2Seq model in [34] to guide the decoder to either directly generate a token from the vocabulary or copy it from the input question. The table schema and the input question are encoded and concatenated as the model input. In contrast, the column names are encoded independently from the encoding of questions in [18], which extended the pointer-generator in SQL generation when the order of conditions in SQL query does not matter. In [20], a unified question-answering framework was proposed to handle ten different natural language processing tasks, including WikiSQL semantic parsing task. To perform question answering on databases with multiple relational tables, there are some other works that aim at guiding the SQL generation indirectly using the answers obtained by query execution [23, 39, 40] or accomplish the goal by directly identifying the correct table cells corresponding to the question answers [10, 31].

Both semantic parsing and language generation approaches show great efficiency in the existing application domains. However, the Question-to-SQL generation task in healthcare domain is still underexplored. There are some efforts in directly seeking answers from unstructured clinical notes to assist doctors with their clinical decision making [16, 41]. However, these problems are significantly different from our task of answering natural language questions on structured EMR data since in our task, the answers to the questions may not be directly included in the structured data. For example, instead of directly retrieving answers, a certain extent of reasoning is required to answer the counting questions starting with "*how many*". There are a few research efforts in solving the Question-to-SQL generation tasks in healthcare domain using semantic parsing and named entity extraction [2, 26]. Due to the domain-specific challenges and the lack of large-scale datasets for model training, there are still several challenges for the Question-to-SQL generation in healthcare. For example, due to the commonly occurring abbreviations of healthcare terminology in EMR data and potential typos in questions, it is possible that the keywords provided in questions are not exactly the same ones used in the EMR data. Therefore, besides predicting the SQL structure and columns, one important task in healthcare is correctly predicting condition values in order to ensure the accuracy of query results for input questions. These challenges motivate us to develop a model that can tackle these issues specifically in healthcare. To train and test our model, we also create the MIMICSQL dataset, which consists of Question-SQL pairs based on MIMIC III dataset. This is the first work that focuses on the Question-to-SQL generation on the healthcare databases with multiple relational tables.

3 MIMICSQL DATASET CREATION

To the best of our knowledge, there is no existing dataset for Question-to-SQL generation task in the healthcare domain. In this section, we provide a detailed illustration of Question-SQL pair generation for Question-to-SQL tasks on EMR data.

3.1 MIMIC III Dataset

To ensure both the public availability of the dataset and the reproducibility of the results for Question-to-SQL generation methods, the widely used Medical Information Mart for Intensive Care III (MIMIC III) dataset [8, 13] is used in this paper to create the Question-SQL pairs. Typically, the healthcare related patient information is grouped into five categories in healthcare literature, including demographics (Demo), laboratory tests (Lab), diagnosis (Diag), procedures (Pro), and prescriptions (Pres). We extracted patient information and prepared a specific table for each category separately. These tables compose a relational patient database where tables are linked through patient ID and admission ID as shown on the top of Figure 1.

3.2 MIMICSQL Generation

Based on the aforementioned five tables, we create the MIMICSQL dataset, including the Question-SQL pairs along with the logical format for slot filling methods, specifically for such Question-to-SQL generation task. Figure 1 provides an overview of basic components used for MIMICSQL generation. Due to the large amount of information included in EMR database, it is challenging and time-consuming for domain experts to manually generate the Question-SQL pairs. It should be noted that, for machine generated questions, there exists some drawbacks, including not being natural compared

to questions provided by humans and usually are not grammatically accurate. In this paper, we take advantage of both human and machine generation to collect the Question-SQL pairs for the MIMICSQL dataset in the following two steps.

3.2.1 Machine Generation of Questions. Following the question types used in [14], there are two types of questions in MIMICSQL, including retrieval questions and reasoning questions. Following the generation of question templates in [22], we first identify the questions that are possibly asked on the EMR data and then normalize them by identifying and replacing the entities regarding table headers, operations, and condition values with generic placeholders. The question templates for retrieval and reasoning questions are finally integrated into two generic templates. These question templates provide a guidance regarding the question topics or perspectives for the machine generated questions.

(1) **Retrieval questions** are designed to directly retrieve specific patient information from tables. The two generic templates mainly used for retrieval questions include:
   - What is the H_1 and H_2 of Patient *Pat* (or Disease *D*, or Procedure *Pro*, or Prescription *Pre*, or Lab test *L*)?
   - List all the Patients (or Disease, or Procedures, or medications, or lab tests) whose H_1 O_1 V_1 and H_2 O_2 V_2.

(2) **Reasoning questions** are designed to indirectly collect patient information by combining different components of five tables. The templates mainly used for reasoning questions include:
   - How many patients whose H_1 O_1 V_1 and H_2 O_2 V_2?
   - What is the maximum (or minimum, or average) H_1 of patient whose H_2 O_2 V_2 and H_3 O_3 V_3?

Here, H_i, O_i, V_i represent placeholders for the i^{th} table column used in the question, its corresponding operation and condition value, respectively. In order to avoid complicated query structure, the number of conditions in each question cannot exceed a predefined threshold, which is set to be 2 in this work.

During question generation, the corresponding SQL query for each question is also generated simultaneously. In order to respond to all questions without changing the query structure and facilitate the prediction of SQL for Question-to-SQL models, we adopt a general **SQL template** SELECT \$AGG_OP (\$AGG_COLUMN)$^+$ FROM \$TABLE WHERE (\$COND_COLUMN \$COND_OP \$COND_VAL)$^+$. Here, the superscript "+" indicates that it allows one or more items. *AGG_OP* is the operation used for the selected AGG_COLUMN and takes one of the five values, including "NULL" (representing no aggregation operation), "COUNT", "MAX", "MIN" and "AVG". *AGG_COLUMN* is the question topic that we are interested in each question and is stored as the column header in tables. Since it is possible for a given question to be related to more than one table, *TABLE* used here can be either a single table or a new table obtained by joining different tables. The part after WHERE represents the various conditions present in the question and each condition takes the form of (\$COND_COLUMN \$COND_OP \$COND_VAL). During query generation, we mainly consider five different condition operations, including "=", ">", "<", ">=" and "<=".

3.2.2 Natural Language Question Collection. These machine generation criteria make it practical to effectively obtain a set of Question-SQL pairs, however, there are two main drawbacks for the machine generated template questions. On the one hand, the questions may not be realistic in the clinical practice. For example, the unreasonable question "*How many patients whose primary disease is newborn and marital status is married?*" will also be generated. On the other hand, the template questions tend to be not as natural as questions asked by doctors since they follow a fixed structure provided in the question templates. In order to overcome these drawbacks, we recruited eight Freelancers with medical domain knowledge on a *crowd-sourcing platform named Freelancer*[1] to filter and paraphrase the template questions in three steps: (1) To ensure that the generated questions are realistic in the healthcare domain, each machine generated question is validated to ignore the unreasonable template questions. (2) Each selected template question is rephrased as its corresponding natural language (NL) question. (3) The rephrased questions are further validated to ensure that they share the same meaning as the original template questions.

Table 1: Statistics of MIMICSQL dataset. The tables are in the order of Demographics, Diagnosis, Procedure, Prescriptions, and Laboratory tests.

Data	Value
# of patients	46,520
# of tables	5
# of columns in tables	23/5/5/7/9
# of Question-SQL pairs	10,000
Average template question length (in words)	18.39
Average NL question length (in words)	16.45
Average SQL query length	21.14
Average aggregation columns	1.10
Average conditions	1.76

3.3 MIMICSQL Statistics

MIMICSQL dataset is publicly available at[2]. We include 10,000 Question-SQL pairs in MIMICSQL whose basic statistics are provided in Figure 2 and Table 1. Figure 2(a) and Figure 2(b) shows the distributions of the question length for template questions and natural language questions, respectively. The distribution of the SQL length is given in Figure 2(c). Figure 2(d) shows the distribution of number of questions over five tables. Note that the total number of questions in Figure 2(d) is more than 10,000 since some questions are related to more than one table.

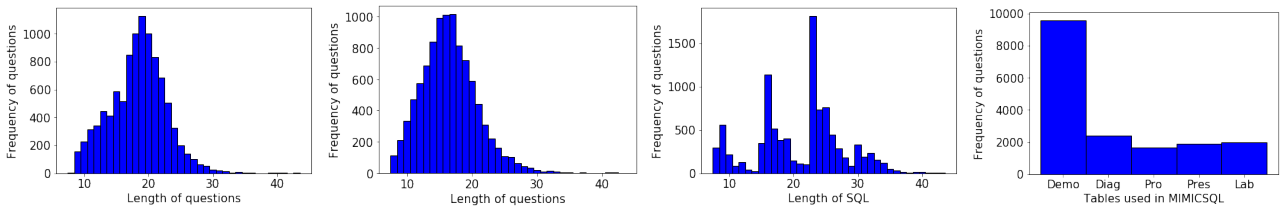
4 A TRANSLATE-EDIT MODEL FOR QUESTION-TO-SQL QUERY GENERATION

In this section, we will first formulate the Question-to-SQL query generation problem. Then, we present our TREQS model in detail.

4.1 Problem Formulation

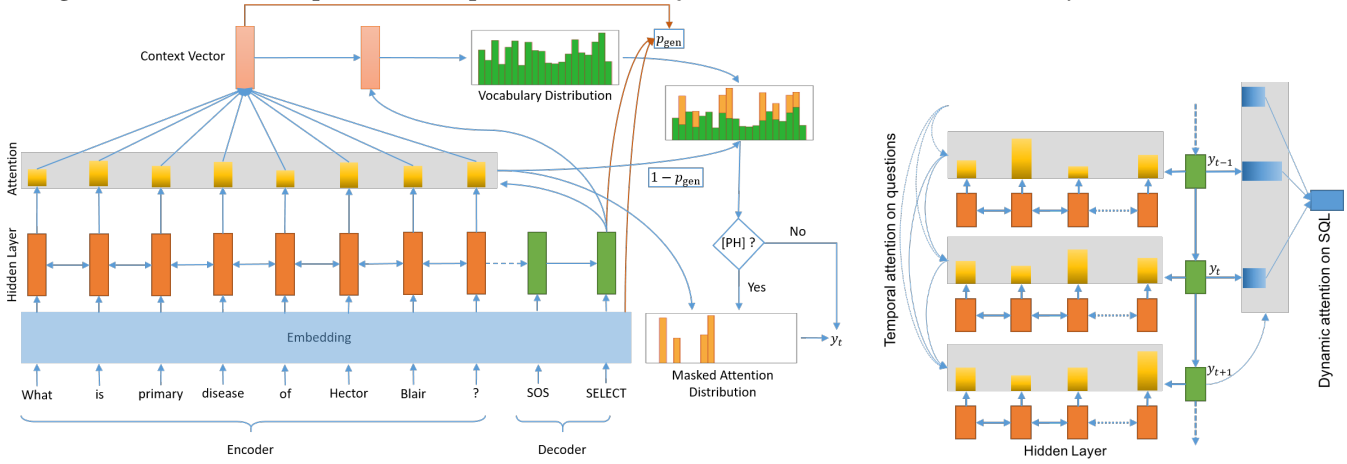
In this paper, we aim to translate healthcare related questions asked by doctors to database queries and then retrieve the answer from health records. We adapt the language generation approach in our model, since questions may be related to a single table or multiple tables, and keywords in the questions may not be accurate due to the healthcare terminology involved. To tackle the challenges for general applications, we propose a translate-edit model that first

[1] www.freelancer.com
[2] https://github.com/wangpinggl/TREQS



(a) Dist. of length of template questions. (b) Dist. of length of NL questions. (c) Dist. of length of SQL queries. (d) Dist. of No. of Questions.

Figure 2: Distribution of questions and queries in MIMICSQL dataset. "Dist." is used as an acronym for "Distribution".



(a) (b)

Figure 3: (a) The overall framework of the proposed TREQS model. [PH] represents the out of vocabulary words in condition values. (b) Illustration of dynamic and temporal attention mechanisms used in TREQS.

generates a query draft using a language generation model and then edits based on the table schema.

Let us denote a given question by $x = (x_1, x_2, ..., x_J)$, the table schema context information as z and the corresponding query as $y = (y_1, y_2, ..., y_T)$, where J and T represents the length of the input and output, respectively. x_j and y_t denote the one-hot representations of the tokens in the question and query, respectively. Then, the goal of our model is to infer y from x based on z with probability $P(y|x, z)$. In our approach, we assume that the table schema information z is implicitly included in the input questions as semantic information. Therefore, during the translation, we only need to deal with inferring y from x. However, since the exact table schema has not appeared at this stage, the generated query can only roughly capture this information. At the second stage, we edit the query draft based on the table schema and look-up tables of content keywords to recover the exact information. This two-stage strategy allows us to easily adapt our model to other general purpose tasks. In the following sections, we will introduce our model layer-by-layer in more detail.

4.2 The Proposed TREQS Model

Now we introduce the details of the three components in the proposed **TR**anslate-**E**dit Model for **Q**uestion-to-**S**QL (**TREQS**) generation. Figure 3(a) shows the framework of the proposed model.

4.2.1 Sequence-to-Sequence Framework. We adopt a RNN sequence-to-sequence (Seq2Seq) framework for the Question-to-SQL generation task. Our Seq2Seq framework is composed of a question encoder (a single-layer bidirectional LSTM [11]) and a SQL decoder (a single-layer unidirectional LSTM). The encoder reads a sequence of word embeddings of input tokens and turns them into a sequence of encoder hidden states (features) $h^e = (h_1^e, h_2^e, \ldots, h_J^e)$, where the superscript e indicates that the hidden states are obtained from the encoder, and $h_j^e = \overrightarrow{h}_j^e \oplus \overleftarrow{h}_{J-j+1}^e$ is the concatenation of the hidden states of forward and backward LSTM. At each decoding step t, the decoder takes the encoder hidden states and word embedding of the previous token as an input and produce a decoder hidden state h_t^d. Both word embeddings in the encoder and decoder are taken from the same matrix W_{emb}. The decoder LSTM hidden and cell states are initialized with

$$\begin{aligned} h_0^d &= \tanh\left(W_{\text{e2dh}}(\overrightarrow{h}_J^e \oplus \overleftarrow{h}_1^e) + b_{\text{e2dh}}\right) \\ c_0^d &= \tanh\left(W_{\text{e2dc}}(\overrightarrow{c}_J^e \oplus \overleftarrow{c}_1^e) + b_{\text{e2dc}}\right) \end{aligned} \tag{1}$$

where the weight matrices W_{e2dh}, W_{e2dc}, and vectors b_{e2dh}, b_{e2dc} are learnable parameters.

4.2.2 Temporal Attention on Question. At each decoding step t, the decoder not only takes its internal hidden state and previously generated token as input, but also selectively focuses on parts of the question that are relevant to the current generation. However, the standard attention models proposed in the literature [1, 19] cannot prevent the decoder from repetitively attending on the same part of the question, therefore, we adopt a temporal attention strategy [21] that was demonstrated to be effective in tackling such problem.

To achieve this goal, we first define an alignment score function between the current decoder hidden state and each of the encoder

hidden states as follows:

$$s^e_{tj} = (h^e_j)^\top W_{\text{align}} h^d_t \tag{2}$$

where W_{align} are parameters. As shown in the left-hand side of Figure 3(b), to avoid repetitive attention, we penalize the tokens that have obtained high attention scores in the previous decoding steps with the following normalization rule:

$$s^{\text{temp}}_{tj} = \begin{cases} \exp(s^e_{tj}) & \text{if } t = 1 \\ \frac{\exp(s^e_{tj})}{\sum_{k=1}^{t-1} \exp(s^e_{kj})} & \text{if } t > 1 \end{cases}, \quad \alpha_{tj} = \frac{s^{\text{temp}}_{tj}}{\sum_{k=1}^{J} s^{\text{temp}}_{tk}} \tag{3}$$

where s^{temp}_{tj} is the new alignment score with temporal dependency, and α_{tj} is an attention weight at current decoding step. With the temporal attention mechanism, we finally obtain a context vector for the input question as follows:

$$z^e_t = \sum_{j=1}^{J} \alpha_{tj} h^e_j. \tag{4}$$

4.2.3 Dynamic Attention on SQL. In our Question-to-SQL generation task, different parts of a query may not strictly have sequential dependency. For example, switching two conditions in a query will yield the same query. However, when generating the condition values, the decoder may need to not only take the previously generated token, its own hidden states and encoder context vector into consideration, but also places more attention on the previously generated table names and headers as shown in the right-hand side of Figure 3(b). Therefore, we introduce a dynamic attention mechanism to the decoder [28, 30], which allows it to dynamically attend on the previous generated tokens.

More formally, for $t > 1$, the alignment scores (denoted by $s^d_{t\tau}$, $\tau \in \{1, ..., t-1\}$) on previously generated tokens can be calculated in the same manner as the alignment scores for the encoder. Then, the attention weight for each token is calculated as follows:

$$\alpha^d_{t\tau} = \frac{\exp(s^d_{t\tau})}{\sum_{k=1}^{t-1} \exp(s^d_{tk})} \tag{5}$$

With the attention distribution and the decoder hidden states, we can calculate the decoder-side context vector as follows:

$$z^d_t = \sum_{\tau=1}^{t-1} \alpha^d_{t\tau} h^d_\tau \tag{6}$$

4.2.4 Controlled Generation and Copying. A Question-to-SQL generation task is very different from the general purpose language generation tasks. First, there are strict templates for SQL queries. For example, SELECT $AGG_OP ($AGG_COLUMN)$^+$ FROM $TABLE WHERE ($COND_COLUMN $COND_OP $COND_VAL)$^+$ is the template we used. Second, the aggregation and condition columns in queries are table headers, which usually do not exactly appear in the questions. For instance, for a given question: "How many patients who have bowel obstruction and stay in hospital for more than 10 days?", its corresponding query looks like "SELECT COUNT (PATIENT_ID) FROM DEMOGRAPHIC WHERE PRIMARY_DISEASE = bowel obstruction AND DAYS_OF_STAY > 10". Obviously, we cannot find words, like PATIENT_ID, PRIMARY_DISEASE, and DAYS_OF_STAY, in the question. Third, the values of conditions should be best possibly retrieved from questions, such as "bowel obstruction" and "10" in the above example, since the questions may contain terms that are out-of-vocabulary (OOV).

Because of these characteristics, our decoder combines a generation network and a pointer network [33] for the token generation. The pointer network has been widely used in language modeling and generation tasks, such as abstractive text summarization [27] and question-answering [20], due to its ability of copying OOV tokens in the source and context sequences to the target sequences. However, in our model, it is primarily used for generating the words in-vocabulary and putting placeholders, denoted as [PH], for OOV words. Intrinsically, it is only used in generating condition values in SQL queries. Formally, to generate a token at step t, we first calculate the probability distribution on a vocabulary $\mathcal{V}$ as follows:

$$\begin{aligned} \tilde{h}^d_t &= W_z(z^e_t \oplus z^d_t \oplus h^d_t) + b_z \\ P_{\mathcal{V},t} &= \text{softmax}\left(W_{\text{emb}}(W_{\text{d2v}}\tilde{h}^d_t + b_{\text{d2v}})\right) \end{aligned} \tag{7}$$

where W_z, W_{d2v}, b_z, and b_{d2v} are parameters. We reuse the syntactic and semantic information contained in the word embedding matrix in token generation. Then, combining with the pointer mechanism, the probability of generating a token y_t is calculated by

$$P(y_t) = p_{\text{gen},t} P_{\text{gen}}(y_t) + (1 - p_{\text{gen},t}) P_{\text{ptr}}(y_t) \tag{8}$$

where the probability $P_{\text{gen}}(y_t)$ given by the generation network is calculated as follows:

$$P_{\text{gen}}(y_t) = \begin{cases} P_{\mathcal{V},t}(y_t) & y_t \in \mathcal{V} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

The probability $P_{\text{ptr}}(y_t)$ by the pointer network is obtained with the following attention distribution

$$P_{\text{ptr}}(y_t) = \begin{cases} \sum_{j:x_j=y_t} \alpha^e_{tj} & y_t \in \mathcal{X} \cap \mathcal{V} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where $\mathcal{X}$ is a set with all tokens in a question. $p_{\text{gen},t}$ is a 'soft-switch' (probability) of using a generation network for token generation

$$p_{\text{gen},t} = \sigma(W_{\text{gen}} z^e_t \oplus h^d_t \oplus E_{y_{t-1}} + b_{\text{gen}}) \tag{11}$$

where $E_{y_{t-1}}$ is the word embedding of the previous token y_{t-1}. W_{gen} and b_{gen} are model parameters. Note that all OOV words in the question have been replaced with the placeholder [PH] for the condition values. In our model, the vocabulary is a union of two sets, i.e., vocabulary of regular tokens and a vocabulary of template keywords as well as table names and headers, denoted as $\mathcal{V}_{\text{schema}}$. Since $\mathcal{X} \cap \mathcal{V}_{\text{schema}} = \emptyset$, the template, table names and headers in a SQL rely only on the generation network. On the other hand, keywords of the condition values and placeholder are obtained from both generation and pointer networks. Note that we always switch the option of [PH] in Figure 3(a) to"No" during training.

With the final probability of generating a token y_t, we are ready to define our loss function. In this paper, we adopt the cross-entropy loss which tries to maximize the log-likelihood of observed sequences (ground-truth), i.e.,

$$\mathcal{L} = -\log P_\theta(\hat{y}|x) = \sum_{t=1}^{T} \log P_\theta(\hat{y}_t|\hat{y}_{<t}, x) \tag{12}$$

where θ denotes all the model parameters, including weight matrices W and biases b. $\hat{y} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_T)$ represents a ground-truth SQL sequence in the training data and $\hat{y}_{<t} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_{t-1})$.

4.2.5 Placeholder Replacement. After a query has been generated, we replace each [PH] with a token in the source question. For a [PH] at time step t', the replacement probability is calculated by

$$P_{\text{rps}}(y_{t'}) = \begin{cases} \sum_{j:x_j=y_{t'}} \alpha^e_{t'j} & y_{t'} \in \mathcal{X} - \mathcal{V} \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

Here, we implement this technique by applying a mask (0 or 1) on the attention weights (named as masked attention mechanism). This replacement technique can make use of the semantic relationships (captured by attention and decoder LSTM) between previously generated words and their neighboring OOV words. Intuitively, if the model attends word x_j at the step $t-1$, it has a high chance of attending the neighboring words of x_j at step t. This meta-algorithm can be used for any attention-based Seq2Seq model.

4.2.6 Recover Condition Values with Table Content. So far, we have used our translate-edit model to translate given questions on a table to the SQL queries without explicitly using any table content and schema. However, we cannot guarantee that all these queries are executable, since the condition values in the questions may not be accurate. In the aforementioned example, the doctor may ask "How many patients who have **bowel obstruct** and stay in hospital for more than 10 days?", then, one of the conditions in the SQL is "PRIMARY_DISEASE = **bowel obstruct**". Obviously, we will get a different answer since **bowel obstruct** does not appear in the database. To alleviate this problem, *we propose a condition value recover technique to retrieve the exact condition values* based on the predicted ones. This approach makes use of string matching metric ROUGE-L [17] (L denotes the longest common sub-sequence) to find the most similar condition value from the look-up table for each predicted one, and then replaces it. In our implementation, we calculate both word- and character-level similarities, i.e., ROUGE-L scores, between two sequences.

5 EXPERIMENTS

In this section, we first introduce the datasets used in our experiments, and then briefly describe the baseline comparison methods, implementation details, and evaluation metrics. Finally, different sets of qualitative and quantitative results are provided to analyze the query generation performance of the proposed model.

5.1 Experimental Settings

5.1.1 Dataset Description. We use both template and natural language (NL) questions in MIMICSQL dataset (described in Section 3) for evaluation. We first tokenize both source questions and target SQL queries using Spacy package[3]. Then, they are randomly split into training, development and testing sets in the ratio of 0.8/0.1/0.1. To recover the condition values, we also created a look-up table that contains table schema and keywords, i.e., table name, header and keywords of each column. Finally, for template questions in the testing set, we also generated a dataset that has missing information and typos (*testing with noise*) to demonstrate the effectiveness of our condition value recover technique.

5.1.2 Comparison Methods. We demonstrate the superior performance of our TREQS model by comparing it with the following methods. The first two are slot filling methods and generate logic format of queries, while the others produce SQL queries directly.

- **Coarse2Fine model [5]:** It is a two-stage structure-aware neural architecture for semantic parsing. For a given question, a rough sketch of the logical form is first generated by omitting low-level information, such as the arguments and name entities, which will be filled in the second step by considering both the natural language input and the generated sketch.
- **Multi-table SQLNET (M-SQLNET) [36]:** For SQL with multiple conditions, it may have multiple equivalent variants by varying the order of conditions. SQLNET mainly focuses on tackling the unordered property by leveraging the structure-based dependencies in SQL. However, it can only handle questions on a single table under the table-aware assumption. In this paper, we implemented a multi-table version of SQLNET for comparison.
- **Sequence-to-Sequence (Seq2Seq) model [19]:** In this model, there is a bidirectional LSTM encoder and a LSTM decoder. To be consistent with this paper, we adopt the "general" global attention mechanism described in [19]. The placeholder replacement algorithm is also used in the query generation step to tackle the OOV words problem in this model.
- **Pointer-Generator Network (PtrGen) [27]:** The pointing mechanism is primarily used to deal with the OOV words. Therefore, an extended vocabulary of all OOV words in a batch is built at each training step to encourage the copying of low-frequency words in the source questions, which is different from our model. In our pointer network, we encourage the model to either copy tokens related to the condition values or put placeholders.

Note that the proposed condition value recover mechanism can be combined with different models that directly generate SQL queries, therefore, we also apply it to the results obtained from Seq2Seq and PtrGen to boost their performance. However, it is not applicable to Coarse2Fine and M-SQLNET since their predicted condition values have already been in the look-up table.

5.1.3 Implementation Details. We implemented the proposed TREQS model and M-SQLNET with Pytorch [24]. For all language generation models, the dimension of word embeddings and the size of hidden states (both encoder and decoder hidden states) are set to be 128 and 256, respectively. Instead of using pre-trained word embeddings [25], we learn them from scratch. ADAM optimizer [15] with hyper-parameter $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ is adopted to train the model parameters. The learning rate is set to be 0.0005 with a decay for every 2 epochs and gradient clipping is used with a maximum gradient norm of 2.0. During the training, we set the mini-batch size to be 16 in all our experiments and run all models for 20 epochs. The development set is used to determine the best model parameters. During the testing, we implement a beam search algorithm for the SQL generation and the beam size is set to be 5. To build the vocabulary, we keep the words with a minimum frequency of 5 in the training set. Thus, the vocabulary size is 2353 and it is shared between the source question and target SQL. In our experiments, both the source questions and SQL queries are truncated to 30 tokens. The implementation of our proposed TREQS method is made publicly available at[4].

[3] https://spacy.io/

[4] https://github.com/wangpinggl/TREQS

Table 2: The SQL prediction performance results using logic form accuracy (Acc_{LF}) and execution accuracy (Acc_{EX}).

	Template Questions				NL Questions			
Method	**Development**		**Testing**		**Development**		**Testing**	
	Acc_{LF}	Acc_{EX}	Acc_{LF}	Acc_{EX}	Acc_{LF}	Acc_{EX}	Acc_{LF}	Acc_{EX}
Coarse2Fine	0.298	0.321	0.518	0.526	0.217	0.309	0.378	0.496
M-SQLNET	0.258	0.588	0.382	0.603	0.086	0.225	0.142	0.260
Seq2Seq	0.098	0.372	0.160	0.323	0.076	0.112	0.091	0.131
Seq2Seq + recover	0.138	0.429	0.231	0.397	0.092	0.195	0.103	0.173
PtrGen	0.312	0.536	0.372	0.506	0.126	0.174	0.160	0.222
PtrGen + recover	0.442	0.645	0.426	0.554	0.181	0.325	0.180	0.292
TREQS (ours)	0.712	0.803	0.802	0.825	0.451	0.511	0.486	0.556
TREQS + recover	**0.853**	**0.924**	**0.912**	**0.940**	**0.562**	**0.675**	**0.556**	**0.654**

5.1.4 Evaluation Metrics. To evaluate the performance of different Question-to-SQL generation models, we mainly adopt the following two commonly used evaluation metrics [46]. (1) **Execution accuracy** is defined as $Acc_{EX} = N_{EX}/N$, where N denotes the number of Question-SQL pairs in MIMICSQL, and N_{EX} represents the number of generated SQL queries that can result in the correct answers [46]. Note that execution accuracy may include questions that are generated with incorrect SQL queries which lead to correct query results. (2) In order to overcome the disadvantage of execution accuracy, **logic form accuracy** [46], defined as $Acc_{LF} = N_{LF}/N$, is commonly used to analyze the string match between the generated SQL query and the ground truth query. Here, N_{LF} denotes the number of queries that match exactly with the ground truth query.

5.2 Experimental Results

5.2.1 Query Generation Performance. Table 2 provides the quantitative results on both template questions and NL questions for different methods. The best performing methods are highlighted in bold and the second best performing methods are underlined. It can be observed from Table 2 that the Seq2Seq model is the worst performer among all the compared methods due to its poor generating behavior, including factual errors, repetitions and OOV words. PtrGen performs significantly better than Seq2Seq model since it is able to copy words from the input sequence to the target SQL. As seen from the results, it can capture the factual information and handle OOV words more efficiently. It works well when most words in the target sequence are copied from the source sequence, which is similar to other problems such as abstractive text summarization task [7, 27]. However, in Question-to-SQL task, most tokens (template, table names and headers) are obtained from generation and only condition values are copied from questions to queries. Therefore, the task discourages copying in general, which causes PtrGen model to produce the condition values by generation instead of copying, thus increasing the chances of making mistakes. Coarse2Fine achieves outstanding performance for the questions on a single table. The limitation of Coarse2Fine is that it cannot handle complex SQL generation, such as queries including multiple tables. However, it still outperforms both Seq2Seq and PtrGen in most of the cases. Compared to Coarse2Fine, the M-SQLNET method considers the dependencies between slots using a dependency graph determined by the intrinsic structure of SQL. It performs significantly better than Seq2Seq and PtrGen on both testing and testing with noise set (in Table 3). It also significantly outperforms Coarse2Fine based on the execution accuracy. Compared to all the aforementioned baseline methods, our proposed TREQS model gains a significant performance improvement on both development and testing dataset and 30 percent, on average, more accurate than others.

Table 3: The SQL prediction performance results and their break-down on template testing questions with noise.

Method	**Overall**		**Break-down**					
	Acc_{LF}	Acc_{EX}	Agg_{op}	Agg_{col}	$Table$	Con_{col+op}	Con_{val}	Average
Coarse2Fine	0.444	0.526	0.528	0.528	0.528	0.520	0.444	0.510
M-SQLNET	0.356	0.606	**1.000**	0.953	**0.998**	0.875	0.376	0.840
Seq2Seq	0.157	0.320	0.997	0.862	0.967	0.817	0.206	0.770
Seq2Seq + recover	0.225	0.389	0.999	0.862	0.967	0.817	0.290	0.787
PtrGen	0.301	0.451	0.999	0.988	0.991	0.970	0.309	0.851
PtrGen + recover	0.353	0.498	0.999	0.988	0.991	0.970	0.360	0.862
TREQS (ours)	0.699	0.756	**1.000**	**0.996**	0.995	**0.976**	0.706	0.935
TREQS + recover	**0.872**	**0.907**	**1.000**	**0.996**	0.995	**0.976**	**0.877**	**0.969**

We have also applied the proposed condition value recover technique to three language generation models. It can be observed that such a heuristic approach can significantly boost the performance of these models. From our experiments, we found that language models fail in many cases because they cannot capture all keywords of condition values. As a result, they are not executable or may yield different answers. Hence, the recover mechanism can correct these errors in the conditions of SQL by making the best use of the look-up table. Moreover, as shown in Table 3, after applying some noise to the template testing questions by removing partial condition values or using abbreviations of words, the performance of different models drops. Our TREQS model is affected significantly because it strongly relies on the pointing mechanism to copy keywords of condition values from questions to queries. However, as we can see, the recover mechanism can still correct most of the errors, thus improving the accuracy by more than 20%, which is 13% for the testing set without introducing noise.

5.2.2 Break-down Generation Performance. In order to further evaluate the performance on each component of SQL query, in Tables 3, 4 and 5, we provide the break-down accuracy results based on SQL query structure, including aggregation operation (Agg_{op}), aggregation column (Agg_{col}), table ($Table$), condition column along with its operation (Con_{col+op}), and condition value (Con_{val}). The results of Coarse2Fine are not provided due to its table-aware assumption and its inability in handling multi-table questions. We can observe that there is no significant difference between these methods on predictions of both aggregation operation and table. Seq2Seq model performs relatively worse on aggregation column and condition column and its operation.

It is easy to observe from Tables 2 to 5 that the performance of condition value dominates the overall SQL generation performance. Seq2Seq is not able to capture the correct condition values due to its limitation in handling the OOV words. PtrGen performs slightly better since it is able to copy OOV words directly from the input questions, however, it still cannot capture the condition values as accurately as our proposed TREQS model. We believe that this is due to the fact that we consider temporal attention on questions, dynamic attention on SQL and the controlled generation

Table 4: Accuracy of break-down matching on template questions in MIMICSQL dataset.

Method	Development						Testing					
	Agg_{op}	Agg_{col}	$Table$	Con_{col+op}	Con_{val}	Average	Agg_{op}	Agg_{col}	$Table$	Con_{col+op}	Con_{val}	Average
Coarse2Fine	0.321	0.321	0.321	0.321	0.298	0.316	0.528	0.528	0.528	0.520	0.518	0.524
M-SQLNet	**1.000**	0.978	0.994	0.876	0.274	0.824	**1.000**	0.956	**0.996**	0.881	0.401	0.847
Seq2Seq	0.999	0.950	0.972	0.761	0.119	0.760	0.999	0.865	0.963	0.818	0.210	0.771
Seq2Seq + recover	0.999	0.950	0.972	0.761	0.163	0.769	0.999	0.865	0.963	0.818	0.296	0.788
PtrGen	0.999	0.991	0.992	**0.979**	0.325	0.857	**1.000**	0.988	0.992	**0.985**	0.381	0.869
PtrGen + recover	0.999	0.991	0.992	**0.979**	0.449	0.882	**1.000**	0.988	0.992	**0.985**	0.433	0.880
TREQS (ours)	**1.000**	**0.999**	**0.995**	0.924	0.719	0.927	**1.000**	0.995	**0.996**	0.980	0.810	0.956
TREQS + recover	**1.000**	**0.999**	**0.995**	0.924	**0.859**	**0.955**	**1.000**	**0.996**	**0.996**	0.984	**0.918**	**0.979**

Table 5: Accuracy of break-down matching on NL questions in MIMICSQL dataset.

Method	Development						Testing					
	Agg_{op}	Agg_{col}	$Table$	Con_{col+op}	Con_{val}	Average	Agg_{op}	Agg_{col}	$Table$	Con_{col+op}	Con_{val}	Average
Coarse2Fine	0.319	0.313	0.321	0.260	0.214	0.285	0.524	0.490	0.528	0.448	0.413	0.481
M-SQLNet	**0.994**	**0.939**	0.933	0.722	0.080	0.734	0.989	**0.873**	**0.941**	0.749	0.140	0.738
Seq2Seq	0.978	0.872	0.926	0.466	0.137	0.676	0.970	0.696	0.892	0.563	0.239	0.672
Seq2Seq + recover	0.978	0.872	0.926	0.471	0.174	0.684	0.970	0.696	0.892	0.565	0.296	0.684
PtrGen	0.987	0.917	**0.944**	0.795	0.172	0.766	0.987	0.830	0.926	0.824	0.214	0.757
PtrGen + recover	0.987	0.917	**0.944**	0.795	0.236	0.776	0.987	0.830	0.926	0.824	0.235	0.760
TREQS (ours)	0.990	0.912	0.942	**0.834**	**0.574**	**0.850**	**0.993**	0.827	**0.941**	0.841	0.679	0.856
TREQS + recover	0.990	0.912	0.942	**0.834**	**0.694**	**0.873**	**0.993**	0.827	**0.941**	**0.844**	**0.763**	**0.874**

Table 6: SQL Queries generated by different models on two NL questions in testing set. The incorrectly predicted words are highlighted in red color.

Method	Example 1	Example 2
Question	how many female patients underwent the procedure of abdomen artery incision?	how many patients admitted in emergency were tested for ferritin?
Ground truth	**select** count (distinct demographic."subject_id") **from** demographic inner join procedures on demographic.hadm_id = procedures.hadm_id **where** demographic."gender" = "f" and procedures."short_title" = "abdomen artery incision"	**select** count (distinct demographic."subject_id") **from** demographic inner join lab on demographic.hadm_id = lab.hadm_id **where** demographic."admission_type" = "emergency" and lab."label" = "ferritin"
M-SQLNET	**select** count (distinct demographic."subject_id") **from** demographic inner join procedures on demographic.hadm_id = procedures.hadm_id **where** demographic."gender" = "f" and procedures."short_title" = "parent infus nutrit sub"	**select** count (distinct demographic."subject_id") **from** demographic inner join lab on demographic.hadm_id = lab.hadm_id **where** demographic."admission_type" = "emergency" and lab."label" = "po2"
Seq2Seq	**select** count (distinct demographic."subject_id") **from** demographic inner join procedures on demographic.hadm_id = procedures.hadm_id **where** demographic."gender" = "m" and procedures."long_title" = "other abdomen"	**select** count (distinct demographic."subject_id") **from** demographic inner join lab on demographic.hadm_id = lab.hadm_id **where** demographic."admission_location" = "phys referral/normal deli" and lab."itemid" = "ferritin"
Seq2Seq+recover	**select** count (distinct demographic."subject_id") **from** demographic inner join procedures on demographic.hadm_id = procedures.hadm_id **where** demographic."gender" = "m" and procedures."long_title" = "other bronchoscopy"	**select** count (distinct demographic."subject_id") **from** demographic inner join lab on demographic.hadm_id = lab.hadm_id **where** demographic."admission_location" = "phys referral/normal deli" and lab."itemid" = "51200"
PtrGen	**select** count (distinct demographic."subject_id") **from** demographic inner join procedures on demographic.hadm_id = procedures.hadm_id **where** demographic."gender" = "f" and procedures."long_title" = "spinal abdomen artery"	**select** count (distinct demographic."subject_id") **from** demographic inner join lab on demographic.hadm_id = lab.hadm_id **where** demographic."admission_type" = "emergency" and lab."label" = "troponin i"
PtrGen+recover	**select** count (distinct demographic."subject_id") **from** demographic inner join procedures on demographic.hadm_id = procedures.hadm_id **where** demographic."gender" = "f" and procedures."long_title" = "spinal tap"	**select** count (distinct demographic."subject_id") **from** demographic inner join lab on demographic.hadm_id = lab.hadm_id **where** demographic."admission_type" = "emergency" and lab."label" = "troponin i"
TREQS	**select** count (distinct demographic."subject_id") **from** demographic inner join procedures on demographic.hadm_id = procedures.hadm_id **where** demographic."gender" = "f" and procedures."short_title" = "abdomen artery abdomen"	**select** count (distinct demographic."subject_id") **from** demographic inner join lab on demographic.hadm_id = lab.hadm_id **where** demographic."admission_type" = "emergency" and lab."label" = "ferritin"
TREQS + recover	**select** count (distinct demographic."subject_id") **from** demographic inner join procedures on demographic.hadm_id = procedures.hadm_id **where** demographic."gender" = "f" and procedures."short_title" = "abdomen artery incision"	**select** count (distinct demographic."subject_id") **from** demographic inner join lab on demographic.hadm_id = lab.hadm_id **where** demographic."admission_type" = "emergency" and lab."label" = "ferritin"

Table 7: Visualization of the accumulated attention on conditions that are used in the proposed TREQS approach on NL questions. Different conditions are labeled with different colors. An intense shade on a word indicates a higher attention weight.

Question	Generated SQL
provide the number of patients less than 83 years of age who were diagnosed with pneumococcal pneumonia .	select count (distinct demographic.subject_id) from demographic inner join diagnoses on demographic.hadm_id = diagnoses.hadm_id where demographic.age < 83 and diagnoses.short_title = pneumococcal pneumonia
provide the number of patients whose diagnoses is lap surg convert to open and they are female .	select count (distinct demographic.subject_id) from demographic inner join diagnoses on demographic.hadm_id = diagnoses.hadm_id where demographic.gender = f and diagnoses.long_title = lap surg convert to open they
among patients treated with amitriptyline , calculate the number of female patients .	select count (distinct demographic.subject_id) from demographic inner join prescriptions on demographic.hadm_id = prescriptions.hadm_id where demographic.gender = f and prescriptions.drug = amitriptyline
get me the number of elective hospital admission patients who had coronary artery primary disease .	select count (distinct demographic.subject_id) from demographic where demographic.admission_type = elective and demographic.diagnosis = coronary artery disease
give the number of patients whose admission type is elective and primary disease is abdominal abscess .	select count (distinct demographic.subject_id) from demographic where demographic.admission_type = elective and demographic.diagnosis = abdominal abscess
how many patients aged below 36 years have stayed in the hospital for more than 14 days ?	select count (distinct demographic.subject_id) from demographic where demographic.age < 36 and demographic.days_stay > 14
what is the number of patients whose admission location is emergency room admit and with primary disease t5 fracture ?	select count (distinct demographic.subject_id) from demographic where demographic.admission_location = emergency room admit and demographic.diagnosis = t5 fracture

and copying techniques in the proposed model. We can also observe that the proposed recover technique on the condition values can also improve the model performance significantly on both template questions and NL questions. As shown in Table 3, the condition values can also be recovered effectively even if only partial condition value information is provided in the input questions. More analysis about the recover technique will be provided.

5.2.3 Analysis of the Generated SQL Query. In addition to the quantitative evaluations, we have also conducted an extensive set of qualitative case studies to compare the SQL queries produced by various models. Two examples on NL questions are provided in Table 6. For both examples, different comparison models have given correct answers for the template, table name, and columns. Note that the Coarse2Fine model cannot handle questions on two tables. For example 1, M-SQLNET provides a wrong procedure short title "*parent infus nutrit sub*" due to the mis-classification error. Seq2Seq and Ptr generate a partially correct procedure short title "*other abdomen*" and "*spinal abdomen artery*", respectively. In addition to their inability to obtain the correct condition values, these baseline methods do not even have the ability to correctly predict the condition column "*procedures.short_title*" in example 1. Similarly, in example 2, the generated SQL query by Seq2Seq model is not executable even if it correctly predicts the value "*ferritin*" for the second condition, since it predicts an incorrect condition column "*lab.itemid*". In this case, the recover technique can only recover the condition value "*51200*" for "*lab.itemid*" instead of keeping the condition value "*ferritin*" that is correctly generated. These predicted results indicate that successfully recovering the condition values still requires the language generation models to produce correct condition columns and sufficiently relevant keywords. Note that it is unable to recover the condition values for M-SQLNET since its predicted values are already in the look-up table. Different from these baseline methods, our proposed TREQS model is able to generate totally correct SQL queries for both examples even without applying the recover technique. This shows the efficiency of our TREQS method in predicting the correct condition values without affecting the performance of other components in the SQL query.

5.2.4 Accumulated Attention Visualization. Visualization of attention weights can help in interpreting the model and explaining experimental results by providing an intuitive view about the relationships between generated tokens and source context, i.e., input questions. In Table 7, we show seven natural language examples with reasoning questions and SQL queries that are generated using the proposed TREQS method. The goal here is to investigate if TREQS is able to successfully detect important keywords in a question when generating conditions in its corresponding SQL query. Therefore, we choose to visualize the accumulated attention weights instead of the weights for each of the generated tokens. For example, for the question "*get me the number of elective hospital admission patients who had coronary artery primary disease*", the model mainly focuses on "*elective*" and "*admission*" when generating condition "*demographic.admission_type = elective*", and on "*coronary artery*" when generating "*demographic.diagnosis = coronary artery disease*". In this example, the condition values are mainly obtained by directly copying from the input question since they are explicitly included. On the other hand, the condition value "*f*" in the SQL query for question "*among patients treated with amitriptyline, calculate the number of female patients*" is mainly obtained through the controlled generation and copying technique since "*f*" is not explicitly provided in the input question. Similarly, TREQS model is able to capture relevant keywords for each condition in other examples.

6 CONCLUSION

Large amounts of EMR data are collected and stored in relational databases at many clinical centers. Effective usage of the EMR data, such as retrieving patient information, can assist doctors in making future clinical decisions. Recently, the Question-to-SQL

generation methods have received a great deal of attention due to their ability to predict SQL query for a given question about a database. Such an automated query generation from a natural language question is a challenging problem in the healthcare domain. In this paper, based on the publicly available MIMIC III dataset, a Question-SQL pair dataset (MIMICSQL) is first created specifically for the Question-to-SQL generation task in healthcare. We further proposed a TRanslate-Edit Model for Question-to-SQL (TREQS) generation task on MIMICSQL by first generating the targeted SQL directly and then editing with both attentive-copying mechanism and a recover technique. The proposed model is able to handle the unique challenges in healthcare and is robust to randomly asked questions. Both the qualitative and quantitative results demonstrate the effectiveness of our proposed method.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation grants IIS-1619028, IIS-1707498 and IIS-1838730.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 4th International Conference on Learning Representations*.

[2] Asma Ben Abacha and Pierre Zweigenbaum. 2012. Medical question answering: translating medical questions into sparql queries. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*. ACM, 41–50.

[3] Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 4560–4565.

[4] Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. 33–43.

[5] Li Dong and Mirella Lapata. 2018. Coarse-to-Fine Decoding for Neural Semantic Parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. 731–742.

[6] Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 351–360.

[7] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. 2018. Bottom-Up Abstractive Summarization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 4098–4109.

[8] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. 2000. Physiobank, physiotoolkit, and physionet. *Circulation* 101, 23 (2000), e215–e220.

[9] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *ACL*.

[10] Tong Guo and Huilin Gao. 2019. Table2answer: Read the database and answer without SQL. *arXiv preprint arXiv:1902.04260* (2019).

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[12] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the ACL*. 963–973.

[13] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. MIMIC-III, a freely accessible critical care database. *Scientific data* 3 (2016).

[14] Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. 2018. DVQA: Understanding Data Visualizations via Question Answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5648–5656.

[15] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations* (2015).

[16] Minsuk Lee, James Cimino, Hai Ran Zhu, Carl Sable, Vijay Shanker, John Ely, and Hong Yu. 2006. Beyond information retrieval—medical question answering. In *AMIA annual symposium proceedings*, Vol. 2006. American Medical Informatics Association, 469.

[17] Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. *Text Summarization Branches Out* (2004).

[18] Denis Lukovnikov, Nilesh Chakraborty, Jens Lehmann, and Asja Fischer. 2018. Translating Natural Language to SQL using Pointer-Generator Networks and How Decoding Order Matters. *arXiv preprint arXiv:1811.05303* (2018).

[19] Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 1412–1421.

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

[21] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Ça glar Gulçehre, and Bing Xiang. 2016. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning* (2016), 280–290.

[22] Anusri Pampari, Preethi Raghavan, Jennifer Liang, and Jian Peng. 2018. emrQA: A Large Corpus for Question Answering on Electronic Medical Records. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2357–2368.

[23] Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*. 1470–1480.

[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. *Proceedings of the 31st Conference on Neural Information Processing Systems* (2017), 1–43.

[25] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing*. 1532–1543.

[26] Kirk Roberts and Braja Gopal Patra. 2017. A Semantic Parsing Method for Mapping Clinical Questions to Logical Forms. In *AMIA Annual Symposium Proceedings*, Vol. 2017. American Medical Informatics Association, 1478–1487.

[27] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. 1073–1083.

[28] Tian Shi, Yaser Keneshloo, Naren Ramakrishnan, and Chandan K Reddy. 2018. Neural Abstractive Text Summarization with Sequence-to-Sequence Models. *arXiv preprint arXiv:1812.02303* (2018).

[29] Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. 2018. IncSQL: Training Incremental Text-to-SQL Parsers with Non-Deterministic Oracles. *arXiv preprint arXiv:1809.05054* (2018).

[30] Tian Shi, Ping Wang, and Chandan K Reddy. 2019. LeafNATS: An Open-Source Toolkit and Live Demo System for Neural Abstractive Text Summarization. In *Proceedings of NAACL*. 66–71.

[31] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 771–782.

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[33] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*. 2692–2700.

[34] Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. 2018. Pointing Out SQL Queries From Text. *Technical report.* (2018).

[35] Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100* (2018).

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

[37] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–26.

[38] Semih Yavuz, Izzeddin Gur, Yu Su, and Xifeng Yan. 2018. What It Takes to Achieve 100 Percent Condition Accuracy on WikiSQL. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 1702–1711.

[39] Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. *Proceedings of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (2015), 1321–1331.

[40] Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2016. Neural enquirer: Learning to query tables with natural language. *Proceedings of 2016 NAACL Human-Computer Question Answering Workshop* (2016), 29–35.

[41] Hong Yu, Minsuk Lee, David Kaufman, John Ely, Jerome A Osheroff, George Hripcsak, and James Cimino. 2007. Development, implementation, and a cognitive evaluation of a definitional question answering system for physicians. *Journal of biomedical informatics* 40, 3 (2007), 236–251.

[42] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation. In *Proceedings of NAACL-HLT 2018*. 588–594.

[43] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. In *EMNLP*.

[44] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of EMNLP*. 3911—-3921.

[45] Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 5341–5352.

[46] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv preprint arXiv:1709.00103* (2017).