# Natural Language Interface for Patients' Electronic Health Records

**Revanth Korrapolu**
rrk69

**Madhumitha Sivaraj**
ms2407

## Abstract

Electronic Health Record (EHR) related problems have proved to be costly for hospitals, stress-inducing for physicians, and life-threatening for patients. To solve this, we developed a Natural Language Interface (NLI) that can assist doctors with navigating the maze of patient data. By simply taking in the doctor's question as input, our system will query the patient health database and find relevant information, and then construct a natural language answer as output. Motivated by the TRanslate-Edit Model for Question-to-SQL generative model, we integrate this to translate the input question to a SQL query using a Seq2Seq-based model and edit the generated query. Our application is a case study of the real-world efficacy of using modern natural language understanding and natural language generation techniques to create an EHR NLI. With an accuracy of 55%, our results have shown that current Natural Language Question to SQL models are not accurate enough to made into a practical NLI.

## 1 Introduction

Navigating patient data within Electronic Health Record systems has resulted in physician burnout, according to several studies. A notable AHRQ-funded study (Minimizing Error, Maximizing Outcome) questioned 170,000 physicians and found that more than half of the physicians reported experiencing time pressures when conducting physical examinations. In 2012, a bill passed to push hospitals to use electronic health records, as opposed to the existing paper-based system. The hope was to alleviate some of the burdens and free up time for doctors. Instead, this leads to more misdiagnoses, misperceptions, and overall medical malpractice.

A 2019 study published in Health Affairs demonstrated how easy it is to make mistakes when performing basic tasks (ie. picking a drug from long drop-down menus) on the nation's two leading EHR systems. The researchers found that in roughly 1 out of 1,000 orders, physicians accidentally select the wrong drug and roughly 1 in 5 of those could have resulted in patient harm.[4] In a separate study, Dr. Martin Makary, a surgical oncologist at Johns Hopkins, identified the "poor interface design" as the third-leading cause of death in America. Given the health care environment often consists of long working days, stressful speed, time pressure, and emotional intensity, the atmosphere tends to place doctors and other clinicians at a high risk of burnout. Hence, it is important to streamline processes to help reduce physician burnout and minimize patient risk and misdiagnoses.

There several reasons why this problem is difficult to solve. One reason can be attributed to the lack of medical datasets. To start, there are complex medical ontologies that must be mapped to and from natural language. One example of a medical ontology is the ICD-10 medical codes that provide a unique identifier for medical conditions. Even still, the wide use of abbreviations of medical terminology and data entry errors make it difficult to match keywords in questions to those in the database schema. Furthermore, condition value parsing remains a challenging problem. This entails identify and extracting information for the "WHERE" (condition) clause that consists of one or more conditions. Newer research has looked into even more complex problems, such as the usage for nested questions to nested queries but this challenge was outside the scope of our project. Lastly, the Seq2SQL model has been solved, but the TRanslate-Edit Model for Question-to-SQL model was only published recently, within the past year. This was due to the creation of the MIMIC-SQL data which used crowd-sourcing to aggregate physician question to SQL pairs. In general patient data must be secured under HIPAA compliance

rules, making real-world data fairly difficult to access. This project would not have been possible without the availability of a large-scale datasets like MIMIC-SQL.

In order to combat physician burnout and minimize EHR-related errors, we seek to build a natural language interface (NLI) in which physicians can interact with with EHRs. Our project served as a case study into the efficacy of cutting-edge NLP models for the creation of a Physician NLI. Previously proposed solutions include chat bots which serve to help doctors but there has never been a NLI designed to address this problem. In theory, an NLI would provide a novel solution to an important problem. However, there are many challenges. To start, the NLP tasks to achieve even basic functionality have only been recently developed.

Our approach entails an end-to-end solution that consists of two parts: Natural Language Understanding and Natural Language Generation. To test for the real-world efficacy of a physician NLI, we chose the TRanslate-Edit Model for Question-to-SQL for the NLU task and RosaeNLG for the NLG task. One limitation of the model is its inability to parse complex or nested questions. Additionally, another limitation of our architecture is the inability to retrieve multi-modal data. New research has allowed users to query for images, graphs, or pdfs. One real-world use case would be physicians looking to retrieve a patient's x-ray file. For the scope of our project, we only looked at retrieving structure data within a relational database. Additionally, our NLG implementation will return all answers in natural language. However, for population health management, most queries result in large query outputs which are not ideal when the same information could be represented more succinctly in a table or graph.

### 1.1 Summary of Contributions

Revanth Korrapolu was responsible for developing the Natural Language Understanding portion, while Madhu Sivaraj was in charge of building the Natural Language Generation component.

## 2 Related Work

We were inspired by the TRanslate-Edit Model for Question-to-SQL (TREQS) model as detailed in "Text-to-SQL Generation for Question Answering on Electronic Medical Records" (Wang et al. 2020). TRESQ is a two-stage generation model that boosts Seq2SQL in three parts. First, it translates an input question to a SQL query using a Seq2Seq based model. Then, it edits the generated query with an attentive-copying mechanism. Lastly, it further edits it with task-specific look-up tables.

| 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" |

Figure 1: Comparative analysis of that model with other baseline models (Wang et al, 2020)

This previous figure is from Wang et al.'s paper and presents a comparative analysis of the TREQS generation model with other baseline models (Wang et al. 2020). The top shows the question and ground truth SQL query. The red text shows the inconsistencies in the baseline models. As mentioned earlier, one limitation of the model is its inability to parse complex or nested questions.

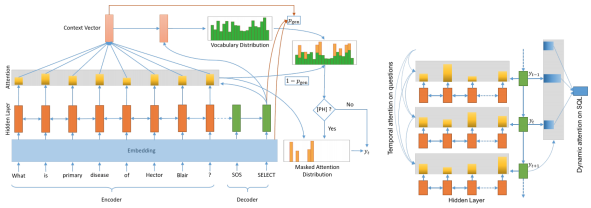


Figure 2: The overall framework of the proposed TREQS model is pictured on the left. [PH] represents the out of vocabulary words in condition values. On the right is an illustration of dynamic and temporal attention mechanisms used in TREQS.

## 3 Overview

Our goal is to create a natural language interface that provides doctors a more intuitive way of interacting patient data and helps alleviate physician burnout in the healthcare industry. Our natural language interface consists of two portions: Natural Language Understanding (NLU) and Natural Language Generation (NLG).

### 3.1 Workflow

A typical workflow starts with a physician asking the interface a natural language question. This question is translated into a SQL query. Assuming that the information can be found in the database, the query will then be run against our patient database. The FHIR database will contain all patient information including the patient id, age, height, weight, symptoms, diagnosis, notes for check-ins, and more. The resulting query response is then formatted into a sentence using natural language generation and provided as output to the client.

### 3.2 Architecture

The interface consist of three layers as shown below in the following figure.

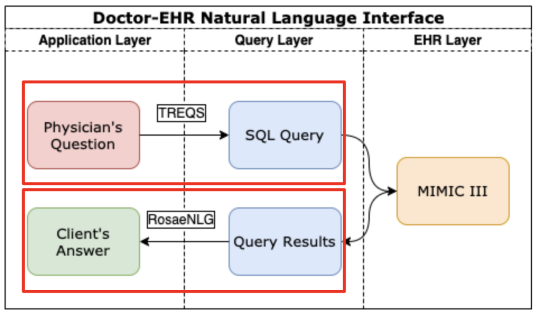


Figure 3: 3-Layer Architecture of Doctor-EHR NLI

This architecture shows three layers which we will implement to simulate a realistic interaction between a doctor and EHR system. Our projects two main components are highlighted in the red boxes. The upper red box defines the NLU task and the lower red box defines the NLG task.

The three layers include:

1. **Application Layer** - defines how the client (doctor) will interact with the interface. This may consist of a web app or chat bot in which the doctor can ask the interface questions about patient data in natural language.

2. **Query layer** - acts an intermediary layer that transposes Physicians' questions into SQL queries (using Seq2SQL) and query results into natural language (using Rosae-NLG).

3. **EHR layer** - consists of our toy database holding patient data. Fast Healthcare Interoperability Resources (FHIR) specifies a new standard of data formatting for healthcare data. As such, we believe a FHIR-compliant database is the archetype of future EHR data systems.

## 4 Implementation

### 4.1 Datasets

We used a large-scale healthcare Question-to-SQL dataset, known as MIMICSQL, by using the publicly available real-world Medical Information Mart for Intensive Care III (MIMIC III) dataset. The MIMIC-III Clinical Database contains tables of clinical data relating to patients who stayed within the intensive care units at Beth Israel Deaconess Medical Center (Johnson et al. 2020). A table is a data storage structure that is similar to a spreadsheet: each column contains consistent information, and each row contains an instantiation of that information. MIMIC-SQL includes 10,000 Question-SQL pairs of doctors' questions and EHR SQL queries.

### 4.2 Natural Language Understanding

| 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 |

Figure 4: Credit: TREQS

The preceding figure displays a table with both the question and the generate SQL for the condition value parsing. For the natural language questions, this visualization shows the accumulated attention weights on the conditions that are used in TREQS. Different conditions are labeled with different colors and an intense shade on a word indicates a higher attention weight.

We utilized the TREQS model in our Natural Language Understanding portion to generate a SQL query using our input question. We run the SQL query against our patient database which contains all patient information, the query response is outputted. This query response will be then used in our natural language generation component.

### 4.3 Natural Language Generation

After the query response is outputted, it is then formatted into a sentence using natural language generation and provided as output to the client. The purpose of NLG in this scope is to generate texts automatically from data.

In order to implement the natural language generation component, we explored various existing tools including Société Générale's Core-NLG and RosaeNLG. We decided on RosaeNLG, an open-source (MIT) NLG library which is easy to use and complete enough to write production grade real life NLG applications. RosaeNLG provides sophisticated linguistic features likes anaphora, verbs, words and adjectives agreement and allows for browser side rendering. RosaeNLG allows us to take the SQL query output in JSON format and render generated sentences. However, one shortcoming with this approach is that the stems of the questions needed to be hard-coded.

## 5 Performance Experiments

The used two forms of accuracy discussed in the TREQS paper:

$$Acc_{EX} = N_{EX}/N$$

1. Execution accuracy - $N$ 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

$$Acc_{LF} = N_{LF}/N$$

2. Logic Form Accuracy - $N_{LF}$ denotes the number of queries that match exactly with the ground truth query. string match between the generated SQL query and the ground truth query

| | Template Questions | | | | NL Questions | | | |
|---|---|---|---|---|---|---|---|---|
| **Method** | **Development** | | **Testing** | | **Development** | | **Testing** | |
| | $Acc_{LF}$ | $Acc_{EX}$ | $Acc_{LF}$ | $Acc_{EX}$ | $Acc_{LF}$ | $Acc_{EX}$ | $Acc_{LF}$ | $Acc_{EX}$ |
| TREQS + recover | **0.853** | **0.924** | **0.912** | **0.940** | **0.562** | **0.675** | **0.556** | **0.654** |

Figure 5

We were not able to re-implement and evaluate all of the other baseline models discussed in the paper, but we were able to corroborate the results for the TREQS model. As shown in figure 5, when TREQS was run on the test NL dataset, the logic form accuracy on the test dataset only proved to be 55% and the execution accuracy was only slightly higher at 65%. These proved to be substantial improvements (+10%) from the baseline models but still too low to function as a viable NLI.

## 6 Conclusion

We developed the first end-to-end medical-domain question answering system which the streamlines the process of navigating poorly designed EHRs. This is made possible by TREQS and RosaeNLG, both of which are limited. With an overall accuracy of 50%, TREQS would not be a viable option in the real world. The accuracy of the TREQS model proved to a be the bottleneck for the overall effectiveness of the NLI. The NLG task was limited, because it could only be accomplished if a valid query out is presented. However, if isolated on its own, the NLG proved to be accurate on structured data, but the output remains syntactically rigid.

## 7 Future Work

Any improvements to the NL Question to SQL task would be the most provide the most improvement to a potential NLI. One avenue of research, would the use of different medical word embeddings, such as BioBERT, when training the Seq2SQL model. This may improve the identification of medical terminologies in the condition extractions.

One current limitation, is that the NLG output is fairly rigid. It takes an strictly, structured input and produces syntactically static output. Better language models could make the output more dynamic wit summarization tasks. It would also be interesting to adapt this model to also handle unstructured or multi-modal databases. Some patient data is stored in pdf's or charts and could still be queried using metadata tags. For example, a doctor may want to see a patients x-ray. This would require the NLG task to also identify when to represent data in different modalities.

Another possible improvement would the implementation of nested question and answering. This is more challenging task but would result in a more intuitive NLI. Nested question and answering has been solved using the original WikiSQL dataset, which Seq2SQL was originally trained on, and could be transferred to the medical domain (TREQS).

## 8 Acknowledgements

We thank Dr. Karl Stratos of the Department of Computer Science, Rutgers University for valuable suggestions and feedback.

## 9 Appendix

Our codebase can be found at:
https://github.com/RevanthK/EHR-NLI

## References


Johnson, A. E. W., Pollard, T. J., Shen, L., Lehman, L. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., & Mark, R. G. (2016) MIMIC-III, a freely accessible critical care database. Scientific Data, 3, 160035.

Wang, P., Shi, T., & Reddy, C.K. (2020) Text-to-SQL Generation for Question Answering on Electronic Medical Records *WWW*. Retrieved April 20, 2020, from http://dmkd.cs.vt.edu/papers/WWW20.pdf

Wang, P., Shi, T., & Reddy, C.K. (2020) TREQS. Retrieved April 20, 2020, https://github.com/wangpinggl/TREQS

Xu, X., Liu, C., & Song, D. (2017) SQLNet: GENERATING STRUCTURED QUERIES FROM NATURAL LANGUAGE WITHOUT REINFORCEMENT LEARNING. Retrieved April 15, 2020, from https://arxiv.org/pdf/1711.04436.pdf

Zhong, V., Xiong, C., & Socher, R. (2017, August 31). Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. Retrieved April 20, 2020, from https://arxiv.org/abs/1709.00103