Generating Querying Code from Text for Multi-Modal Electronic Health Record

Anonymous ACL submission

Abstract

Electronic health records (EHR) contain extensive structured and unstructured data, including tabular information and free-text clinical notes. Querying relevant patient information often requires complex database operations, increasing the workload for clinicians. However, complex table relationships and professional terminology in EHRs limit the query accuracy. In this work, we construct a publicly available dataset, TQGen, that integrates both **T**ables and clinical **T**ext for natural language-to-query **Gen**eration. To address the challenges posed by complex medical terminology and diverse types of questions in EHRs, we designed a medical knowledge module and a questions template matching module. For processing medical text, we introduced the concept of a toolset, which encapsulates the text processing module as a callable tool, thereby improving processing efficiency and flexibility. We conducted extensive experiments to assess the effectiveness of our dataset and workflow, demonstrating their potential to enhance information querying in EHR systems. We will release the project code after our paper is accepted.

1 Introduction

Electronic Health Records (EHR) (Johnson et al., 2016; Pollard et al., 2018; Johnson et al., 2023b) contain a vast amount of tabular and textual information about patients. Retrieving this information often requires complex database queries, posing a challenge for clinicians without specialized database expertise. Converting natural language queries into structured database queries can significantly enhance the efficiency of medical professionals. Previous research has explored table-based text-to-SQL models, including sequence-to-sequence approaches (Dong and Lapata, 2016) and large language model-based methods. With the emergence of large-scale EHR datasets, several works (Raghavan et al., 2021; Lee et al., 2022) have introduced table-based text-to-SQL datasets tailored for EHRs. While these methods have demonstrated promising performance, they still have certain limitations.

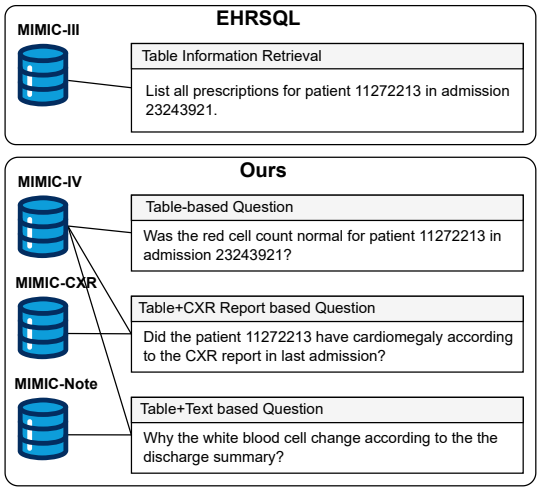

Figure 1: Previous work (such as EHRSQL) focuses only on tabular information, we introduce textual data within tables and leverage multi-modal interactions to create queries.

EHR contains both structured tabular data and unstructured textual information, such as radiology reports and discharge summaries. These texts may be stored directly within table columns as long-form narratives or referenced via external links. Previous studies (Lee et al., 2022) have primarily focused on querying structured tables without integrating text comprehension, see Fig 1 while others (Kweon et al., 2024) have explored long-text processing but lacked the capability to handle multimodal table-text queries in EHRs. Additionally, some works (Bae et al., 2023) have proposed VQA tasks based on tables and chest X-ray (CXR) images. However, in clinical practice, radiology reports are already generated by radiologists, making direct image-based queries less practical. To bridge these gaps, we construct a comprehensive table-text

query generation dataset by integrating structured data from MIMIC-IV (Johnson et al., 2023b), radiology reports from MIMIC-CXR (Johnson et al., 2019), and discharge summaries from MIMIC-Note (Johnson et al., 2023a), facilitating more effective and clinically relevant EHR retrieval.

Besides, EHR contain numerous specialized medical terms, often represented inconsistently (e.g., 'red blood cell' vs. 'RBC'). This variation complicates converting natural language queries into accurate database queries. To address this, we incorporate medical knowledge to identify and map specialized terms in clinician queries to corresponding database terms. Furthermore, the combination of tabular data and text within tables poses challenges for query code generation. To address this, we introduce the concept of a toolset, encapsulating medical text processing functions into callable tools. When the model detects the need to interpret textual data such as CXR reports, it invokes these tools, thereby extending the modality coverage of text-to-SQL systems.

Moreover, the current query statement generation based on electronic health data lacks a standardized processing flow. We propose a query statement processing framework with a large model as the base model, including table content description, medical term matching, question template matching, query statement generation prompts, and code execution inspection operations. These introductions are described in Section 4.

Our contributions are as follows.

1. We have constructed a natural language query dataset that integrates tabular electronic health record (EHR) data with medical text records. This data set expands the textual modality, making natural language queries for EHRs more aligned with real-world scenarios.
2. We propose an EHR query processing framework based on a large language model, incorporating a medical knowledge module, question template matching, and other components to enhance query accuracy. Notably, we introduce the toolset concept and design text processing tools to extend query modality.
3. We evaluated our workflow on the proposed dataset, demonstrating the effectiveness of our approach.

The remainder of the paper is organized into several sections. Section 2 discusses existing related work, Section 3 describes the TQGen dataset generation, Section 4 presents the framework for EHR multi-modal query generation, Section 5 presents the experiments and results, Section 6 discusses the limitation of our work, Section 6 concludes the work and discusses possible future work.

2 Related Work

Classic benchmark datasets such as WikiSQL (Zhong et al., 2017) and Spider 1.0 (Yu et al., 2018) have significantly contributed to text-to-SQL task development. However, with the rise of large language models (LLMs), these datasets have shown limitations, such as lacking domain-specific knowledge and large-scale table structures. Recent benchmarks like DB-GPT-Hub (Zhou et al., 2024) and BIRD (Li et al., 2024) address real-world challenges, including domain-specific knowledge, large-scale tables, and data noise, offering new directions for text-to-SQL research.

In the EHR domain, text-to-SQL tasks focus on extracting information from medical record tables by translating natural language into SQL or other query languages. Wang et al.(Wang et al., 2020) introduced TREQS, which performs text-to-SQL tasks on MIMIC-III. Pampari et al.(Raghavan et al., 2021) developed emrKBQA, a large-scale text-to-logical-form dataset for patient-specific QA on MIMIC-III. Lee et al.(Lee et al., 2022) presented EHRSQL, a text-to-SQL dataset based on MIMIC-III and eICU(Pollard et al., 2018), incorporating time-sensitive and unanswerable queries. EHRNoteQA (Kweon et al., 2024) provides QA tasks from discharge summaries, serving as a long-text benchmark based on MIMIC-IV data.

3 Dataset Construction

Common EHR datasets include eICU-CRD (Pollard et al., 2018), HiRID (Hyland et al., 2020), MIMIC-III (Johnson et al., 2016), and MIMIC-IV (Johnson et al., 2023b), with the MIMIC series being notable for its broad coverage and widespread use. This study utilizes MIMIC-IV, integrating radiology reports from MIMIC-CXR (Johnson et al., 2019) and discharge summaries from MIMIC-Note (Johnson et al., 2023a). These reports are embedded as text or hyperlinks within structured tables, facilitating data association and analysis. A detailed dataset description is provided in Appendix A.1.

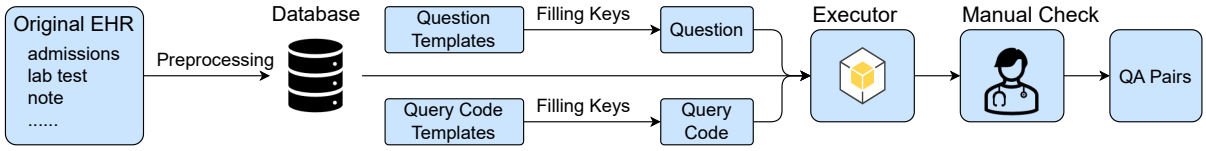

Figure 2: The pipeline of dataset construction. After preprocessing to the EHR data, we create template for question and query code, then execute the code to obatin the question-answer (QA) pairs.

3.1 Preprocessing

In this study, we integrated multi-source datasets by standardizing and linking the raw data using the unique patient identifier (subject_id) and hospital admission ID (hadm_id). Additionally, intensive care unit stay ID (stay_id) and chest X-ray study ID (study_id) were utilized to identify eligible patient cohorts, ensuring a high-quality data foundation for subsequent analyses.

During preprocessing, all table fields underwent type validation and standardization. Text data was converted to lowercase for consistency. To improve query efficiency, related tables were merged (e.g., integrating diagnoses with d_icd_diagnoses). Detailed preprocessing methods are provided in Appendix A.2.

3.2 Question Template

Since the data involves multi-modal information from different modalities (tables, text), we define the scope of question templates using two dimensions: table-based and text-based question.

Table-based questions are associated with structured information from EHR tables. These questions address patient demographics, diagnoses, procedures, medications, and other clinical details typically recorded in a structured EHR format. The dataset offers a rich collection of questions derived from EHR tables, making it a highly valuable resource in this context. We utilized question templates from the MIMIC-III version of EHRSQL, adapting them to align with the MIMIC-IV schema with necessary modifications. Approximately 100 question templates were constructed for table-based queries, with examples provided in Appendix A.2.

The text-based question involves questions derived from discharge summaries and CXR radiology reports. These questions cover patient conditions, changes in the CXR radiology reports, the patient's admission history, discharge diagnoses, and more. We enhanced the templates to handle queries specific to individual patients and for comparisons between two consecutive CXR studies. Approximately 80 question templates were constructed for text-based queries, with examples provided in Appendix A.2.

3.3 Query Answer Generation

In Fig 2, for each question template, we design a corresponding query statement, which is executed after inserting relevant keywords to query results. For questions requiring answer extraction from long-text data, we input both the text and the corresponding question into a large pre-trained model, followed by manual verification of the generated responses. In cases where the query yields no valid information (e.g., inquiries about examinations not performed on a given patient), predefined prompts are used as response outputs.

3.4 Dataset Distribution

We conduct a comparative analysis of previous datasets, as presented in Table 1, and provide statistics on the distribution of question counts across different modalities, as shown in Table 8. Furthermore, we perform a classification analysis based on question complexity, categorizing questions into two levels: Level I for questions with no more than three constraint conditions (eg. subject_id, diagnoses_name), and Level II for those with more than three.

4 Methodology

4.1 Preliminary

In this work, we focus on addressing health-related queries using information from structured EHRs. The reference EHR, denoted as $\mathcal{D} = \{D_0, D_1, ...\}$, D_i represents the i_{th} table in database, and $\mathcal{C}^i = \{C_0^i, C_1^i, ...\}$ corresponds to the column description with in D_i. Given an EHR-based clinical question $q \in Q$, the objective is to extract the final answer by utilizing the information with both $\mathcal{D}$ and $\mathcal{C}$.

We further develop the planning process of LLM as an autonomous agent in EHR question answering. For initialization, the LLM agent is equipped

Dataset	Table	Text	# of Tables	# of Questions
TREQS (Wang et al., 2020)	✓	×	5	10k
EHRSQL (Lee et al., 2022)	✓	×	13.5	24k
EHRXQA (Bae et al., 2023)	✓	×	18	46k
EHRAgent (Shi et al., 2024)	✓	×	10	2k
EHRNote (Kweon et al., 2024)	×	✓	1	1k
TQGen(Ours)	✓	✓	18	12k

Table 1: Dataset comparison with other EHR-based text-to-query dataset.

with a set of pre-built tools $\mathcal{T} = \{T_0, T_1, ...\}$ to interate with the EHR database $\mathcal{D}$. For example, the "SUM", "COUNT" functions can be regarded as tools in SQL querylanguage. The query code generation can be regarded as a combination of tools. Thus we generate the sequence by the following policy: $p_q \sim p(f_1, ...f_t|q, \mathcal{D}, \mathcal{C}, \mathcal{T})$, where $q \in \mathcal{Q}, f_t \in \mathcal{T}$. The final output is obtained by executing the function sequence:

$$y \sim \text{EXECUTOR}(q, f_1, ...f_t, \mathcal{D}, \mathcal{C}, \mathcal{T}) \quad (1)$$

where the EXECUTOR is the query code executor interacting with EHR database.

We then trace the outcome of each interaction back to the LLM agent, which can be either a successful execution result or an error message, to iteratively refine the generated code-based plan. This interactive process is a multi-turn conversation between the planner and executor, which leverages the high-level reasoning capabilities of the LLM to optimize plan refinement and execution.

4.2 Modules

Table Description. In the process of converting natural language into query code, table description is a key module responsible for establishing connections between natural language queries and the structured schema of relational databases. It helps the model accurately map query terms to database columns, tables, or values, thereby improving the generation of query code. We present an example of table description in Appendix A.4.

Matching Module. When parsing a question, the model matches text with values in the table. For example, in "What is the highest red blood count value of patient 01 in admission 02?", table description helps the model locate the subject_id, hadm_id, and label columns in the labtest table to query the patient's value and select the highest one. However, medical terms like red blood count may appear as RBC or red blood cell in different inputs, complicating the mapping process. To address this, we established a standard terminology library to ensure consistent mapping of input terms. Leveraging UMLS (Bodenreider, 2004) standardized medical terms, rbc and red blood cell are uniformly mapped to red blood cell.

To generate queries code, we retrieve historical query templates based on semantic similarity. Pre-stored queries and their corresponding questions are collected and stored. We calculate the similarity between the input question and existing ones using a pre-trained BERT (Reimers and Gurevych, 2019) model. The most similar questions are then retrieved, and their corresponding query templates are extracted. For large template databases, Faiss (Douze et al., 2024) is used for efficient similarity search. If no exact SQL template is found, multiple similar templates are combined to automatically generate the SQL query structure.

Tool Set Module. Since some tables embed links to long texts or directly embed long texts, and the query statement cannot directly extract and understand the corresponding content from the long text, we designed a text understanding tool. When the model parses the input question and finds that the query content involves long texts such as radiology reports or discharge reports, we use the text understanding tool. This tool is packaged into a function. Its input is the long text and the question, and the output is the corresponding value.

In this work, we propose an automatic method for generating dynamic prompts for a text understanding function, Text_Func, based on the original query. For a given question, we extract key entities such as patient ID, admission ID, and medical conditions using a table description module. For example, from the query "Count the number of times that patient 01 had a CXR check indicating effusion in admission 02", we extract patient_id = 01, admission_id = 02, and condition = effusion. Using this extracted information, we dynamically generate a prompt to guide Text_Func in retrieving

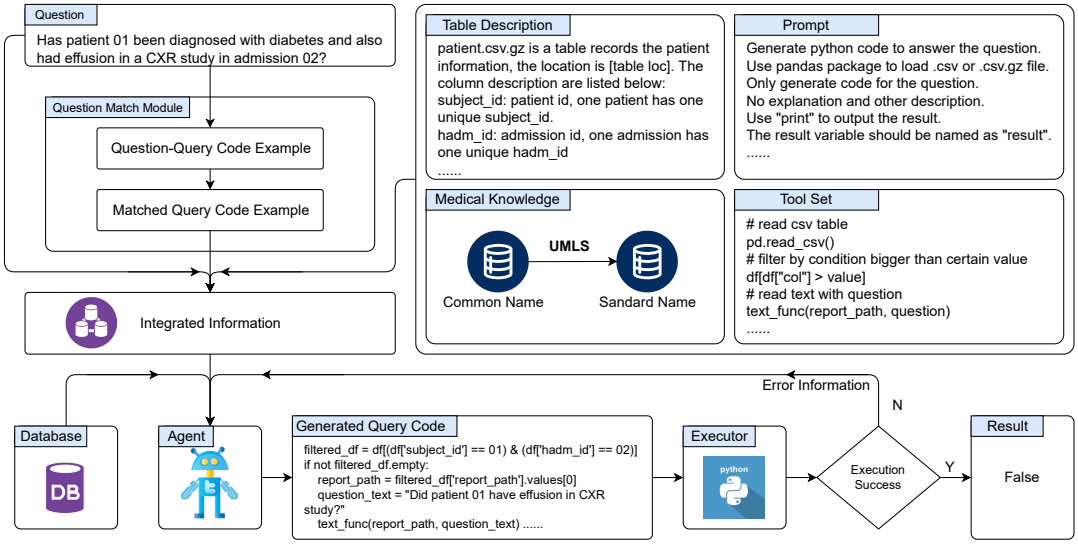

Figure 3: The framework of generating query code from question text. We use python code as example.

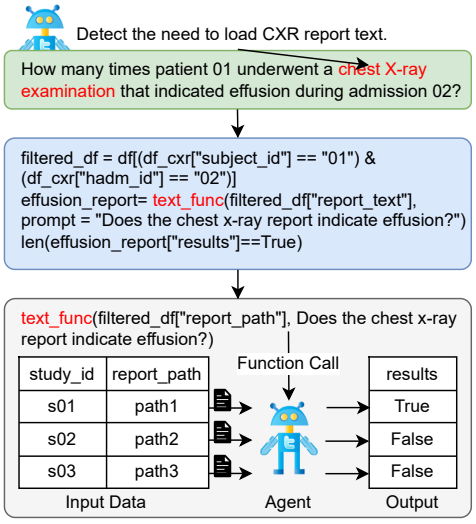

Figure 4: The method to call function to process text.

relevant data from medical records. For instance, the prompt would be: "Does the chest x-ray report of patient 01 in admission 02 indicate effusion?". Figure 4 illustrates the pipeline for using Text_Func to process the text.

Code Inspection Module. The code executor automatically extracts the code from the LLM agent output and executes it within the local environment:

$$O(q) = \text{EXECUTOR}(S(q))) \quad (2)$$

After execution, it sends the results of execution back to the LLM agent for potential plan refinement and further processing.

We observe that the generated query statements do not always execute successfully. To address this issue, we incorporate a repair module to refine queries that fail during execution. When the generated query statement $S(q)$ encounters an error in the executor, we identify potential issues such as incorrect file path references, column mismatches, or erroneous value assignments. To improve query accuracy, we collect the error messages returned by the executor along with the original question, the generated query, guiding prompts, and relevant toolbox resources. This information is then fed back into the LLM agent iteratively until a valid query is produced or the predefined query attempt limit is reached. The equation is as follows:

$$S(q) = \text{LLM}(S(q), \mathcal{I}, \mathcal{T}, q, \text{error\_info}) \quad (3)$$

The logic of the code inspection module can be found in Algorithm 1.

4.3 Evaluation

Exact-Match Accuracy (EM) (Yu et al., 2018). This metric measures whether all SQL components $\mathbb{C} = \{C_k\}$ of the predicted SQL query match the ground-truth SQL query. It can be computed as follows:

$$EM = \frac{\sum_{i=1}^{N} \mathbb{I}\left(\bigwedge_{C_k \in \mathbb{C}} Y_i^{C_k} = \hat{Y}_i^{C_k}\right)}{N} \quad (4)$$

Execution Accuracy (EX) (Yu et al., 2018). This metric evaluates the performance by comparing whether the execution result sets of the ground-truth and predicted SQL queries are identical. It can be computed as:

$$EX = \frac{\sum_{i=1}^{N} \mathbb{I}\left(V_i = \hat{V}_i\right)}{N} \quad (5)$$

where $\mathbb{I}(\cdot)$ is an indicator function that equals 1 if the condition inside is satisfied, and 0 otherwise.

LLM-based Score For long-text answers, such as listing medications or responding to hospitalization reports, the previous metrics are not suitable. Inspired by works LLaVa-Med (Li et al., 2023; Kweon et al., 2024), we use GPT-4 (Achiam et al., 2023) to evaluate the accuracy of model-generated answers. The reference answer is manually created and serves as the upper bound. GPT-4 then evaluates the model's output by comparing it to the reference answer. It then assigns a score on a scale from 1 to 10, where 1 indicates poor accuracy and 10 reflects a highly accurate response.

Algorithm 1 Algorithm Framework

Input: EHR databse $\mathcal{D}$, Input question $q \in \mathcal{Q}$, Column description of EHR $\mathcal{D}$: $\mathcal{C}$, Tool set $\mathcal{T}$, Question samples $\mathcal{Q}_s$, Medical knowledge $\mathcal{M}$, Generation prompt $\mathcal{P}$.
We have guided prompt $\mathcal{I} = [\mathcal{C}, \mathcal{M}, \mathcal{P}]$.

Initialize: try_time : $k \in \{1, \ldots, K\}$, flag $= 0$.
% Match similar question examples
$q_{sim} = \arg \text{TopK}_{max}(sim(q, q_i | q_i \in \mathcal{Q}_s)$
% Generate Query Code
$S(q) = \text{LLM}(\mathcal{I}, \mathcal{T}, q, q_{sim})$
% Loop until max iterations or successful execution
while $k \leq K$ **and** flag $= 0$ **do**
  % Code Execution
  $O(q) = \text{EXECUTOR}(S(q))$
  % Code Check
  if $O(q)$ includes error information **then**
    $S(q) = \text{LLM}(S(q), \mathcal{I}, \mathcal{T}, q, \text{error\_info})$
    $k = k + 1$
  else
    flag $= 1$
  end if
end while
Output: Final answer or output information from $O(q)$

5 Experiment

5.1 Experiment Setup

Task and Datasets. We use test data from our constructed dataset, which includes 1000 Level I and 1000 Level II questions (see Appendix 8). The task is to evaluate the accuracy of the generated query statements and the correctness of the query results. Questions are categorized into two levels based on difficulty: Level I for those with no more than three constraint conditions, and Level II for those with more than three.

Model Select. We used different large models as query generation models, including the Qwen2.5 (Yang et al., 2024) series and the LLaMA (Touvron et al., 2023) series. We also used models with different parameter amounts to measure their impact on query generation capabilities. These models are listed below:

1. Qwen2.5-7B/14B/32B (Yang et al., 2024) is for general-purpose language understanding and generation tasks.
2. Qwen2.5 Code-7B/14B/32B (Hui et al., 2024) is an optimized version of Qwen 2.5 tailored specifically for programming-related tasks.
3. LLaMA 2-7B/13B/34B (Touvron et al., 2023) is for general text understanding task.
4. LLaMA 2 Code 7B/13B/34B (Rozière et al., 2023) is a variant of Llama 2 fine-tuned for coding tasks.

Implementation Details. The experiments were conducted on an NVIDIA GeForce RTX A6000 GPU. To ensure consistency, we set the temperature parameter to 0 during API calls to GPT-4, eliminating randomness in the generated responses. The generated queries are in SQL format, and their execution is facilitated using Python.

5.2 Quantitative Analysis

We evaluated the performance of various models on the dataset using three metrics: exact match accuracy (EM), execution accuracy (EX), and a large language model-based score (LLM-based score). Exact match accuracy and execution accuracy were employed to assess the correctness of results involving simple data types, such as numerical values and strings. In contrast, the LLM-based score was specifically designed to evaluate tasks that involve complex text comprehension and generation.

Table 2 summarizes the experimental results. As shown in the table, with an increase in the model's parameter count, the model's performance on EX, EX, and LLM-based scores improves. Additionally, when the questions are relatively simple, the accuracy of the generated query code and its execution results is higher. However, as the difficulty of the questions increases, the decline in EX for

Model	Size	EM		EX		LLM-based score	
		Level I	Level II	Level I	Level II	Level I	Level II
Qwen 2.5 (Yang et al., 2024)	7b	0.53	0.48	0.80	0.63	7.85	6.04
	14b	0.58	0.53	0.82	0.75	8.53	6.57
	32b	0.61	0.59	0.89	0.80	9.05	7.12
Qwen 2.5 Code (Hui et al., 2024)	7b	0.56	0.52	0.82	0.62	7.79	6.13
	14b	0.60	0.55	0.85	0.76	8.69	6.58
	32b	0.64	0.57	0.91	0.83	9.13	7.23
Llama 2 (Touvron et al., 2023)	7b	0.52	0.48	0.81	0.62	8.02	6.10
	13b	0.55	0.51	0.83	0.73	8.46	6.53
	34b	0.58	0.54	0.90	0.79	8.90	7.20
Llama 2 Code (Rozière et al., 2023)	7b	0.55	0.52	0.81	0.64	7.96	6.16
	13b	0.59	0.54	0.86	0.76	8.45	6.67
	34b	0.63	0.56	0.91	0.81	8.95	7.30

Table 2: Performance comparison of different models on our proposed dataset.

Model	M.K	Q.T.M	C.C	EM		EX		LLM-based score	
				Level I	Level II	Level I	Level II	Level I	Level II
Qwen2.5 14b (Yang et al., 2024)	×	×	×	0.50	0.40	0.72	0.61	7.63	6.10
	✓	×	×	0.54	0.50	0.77	0.66	7.87	6.21
	✓	✓	×	0.58	0.53	0.82	0.70	8.32	6.37
	✓	✓	✓	0.60	0.55	0.85	0.76	8.69	6.58

Table 3: Ablation study of different modules.

the generated query code becomes more significant, while the decrease in EX is less pronounced. This is because complex problems require more function combinations, and although the model uses different function combinations, it ultimately achieves the same result. For text-based query tasks, the LLM-based score also experiences a decline, likely due to the complexity of the questions causing the model to select incorrect texts, thereby affecting accuracy.

We also investigated the differential impact of various functional modules on the overall performance of the dataset. In previous sections, we introduced several key components, including table description, medical knowledge, question template matching, and code checking. For the experimental design, we configured the agent with table descriptions and prompts, and subsequently evaluated the specific influence of the three modules—medical knowledge (M.K), question template matching (Q.T.M), and code checking (C.C)—on the generated query code.

We employed the Qwen2.5-14B (Yang et al., 2024) model as the foundation and randomly sampled 500 instances from a self-constructed test set, encompassing samples with two distinct levels of difficulty. Comparative analysis revealed a significant decrease in the model's accuracy when the M.K, Q.T.M, and C.C modules were disabled.

Disabling the medical knowledge module led to the most significant accuracy decline, likely due to mismatches between query phrases and table entries, causing retrieval failures. The question template matching module also had a notable impact on code generation accuracy, with matched questions and exemplar code boosting performance. However, the model occasionally produced correct code without Q.T.M support. In contrast, the code checking module had a smaller effect, as most SQL queries executed correctly without modification, with adjustments needed only in specific edge cases.

5.3 Case Study

The effectiveness of the aforementioned modules is demonstrated through the experimental results. As illustrated in Fig 5, the impact of different modules on code generation and query result generation is depicted across three subplots: (a) shows the performance differences with and without the

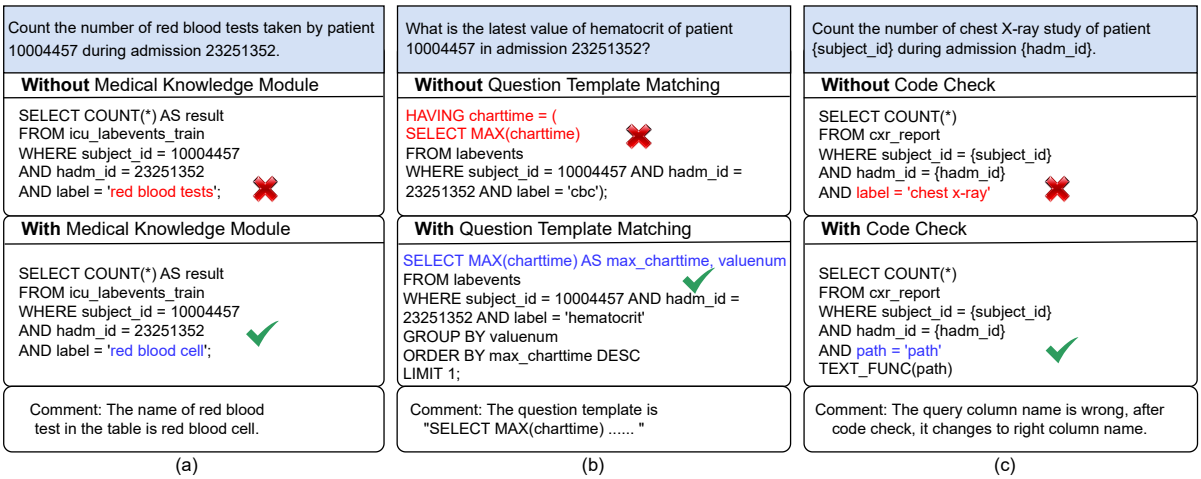

Figure 5: Some examples demonstrate the efficiency of the modules. The top row shows the questions, followed by the generated query codes in the second and third rows — one without the module and the other with the module. The last row explains why the query code is correct.

medical knowledge (M.K) module, indicating a significant improvement in the accuracy of medical term matching when the module is enabled; (b) compares the outcomes with and without the question template matching (Q.T.M) module, highlighting the crucial role of template matching in the task; and (c) validates the contribution of the code checking (C.C) module. The experimental results confirm that all three modules contribute to enhanced accuracy and reliability of the query results.

6 Conclusion

In this work, we present a novel approach for querying EHRs by integrating structured tables and unstructured clinical text. We created a publicly available dataset to facilitate natural language-to-query translation, addressing the complexities of EHR data, including complex table relationships, long-form narratives, and specialized medical terminology. Additionally, we propose a workflow leveraging LLMs, incorporating modules for medical knowledge, question templates, and toolsets to enhance query accuracy. Our findings demonstrate that LLM-powered querying systems can significantly improve EHR data accessibility and usability, paving the way for more efficient clinical information retrieval. Future work will focus on enhancing query accuracy, incorporating multi-modal data sources, and further validating the approach in real-world clinical settings.

Limitation

Despite careful design, our dataset has some limitations. Since it is based on the MIMIC database, its generalizability may be restricted, which could affect the stability, comprehensiveness, and applicability of our model. Future work should address these challenges. While our research represents a significant step in multimodal EHR QA systems, there is still room for improvement. Key future directions include expanding the dataset by enhancing multimodal dialogue systems and integrating mechanisms to handle unanswerable or ambiguous questions, which are crucial for real-world applications. These efforts will leverage our dataset as a valuable resource and lay the foundation for more comprehensive healthcare solutions.

Ethical and Privacy Considerations

In accordance with the PhysioNet Certified Health Data Use Agreement, we strictly prohibit transferring confidential patient data (MIMIC-IV) to third parties, including via online services like APIs. To ensure compliance, we use locally deployed models for testing, preventing third-party access to sensitive patient information. We continuously monitor our adherence to these guidelines and relevant privacy laws to ensure ethical data use. Sensitive information, such as patient names and visit times, has been appropriately processed to protect patient privacy.

Acknowledgments

We thank the MIMIC-IV and MIMIC-IV-Note datasets for providing valuable clinical and textual data that support our research. These resources have been essential in developing and evaluating our multi-modal EHR QA system.

References

OpenAI Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, and Altman et al. 2023. Gpt-4 Technical Report. *arXiv*.

Seongsu Bae, Daeun Kyung, Jaehee Ryu, Eunbyeol Cho, Gyubok Lee, Sunjun Kweon, Jungwoo Oh, Lei Ji, Eric I-Chao Chang, Tackeun Kim, and Edward Choi. 2023. Ehrxqa: A Multi-Modal Question Answering Dataset for Electronic Health Records with Chest X-ray Images. In *Conference on Neural Information Processing Systems (NeurIPS)*. arXiv.

O. Bodenreider. 2004. The Unified Medical Language System (UMLS): integrating biomedical terminology. *Nucleic Acids Research*, 32(90001):267D–270.

Li Dong and Mirella Lapata. 2016. Language to Logical Form with Neural Attention. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Stephanie L. Hyland, Martin Faltys, Matthias Hüser, Xinrui Lyu, Thomas Gumbsch, Cristóbal Esteban, Christian Bock, Max Horn, Michael Moor, Bastian Rieck, Marc Zimmermann, Dean Bodenham, Karsten Borgwardt, Gunnar Rätsch, and Tobias M. Merz. 2020. Early prediction of circulatory failure in the intensive care unit using machine learning. *Nature Medicine*, 26(3):364–373.

Alistair Johnson, Tom Pollard, Steven Horng, Leo Anthony Celi, and Roger Mark. 2023a. Mimic-iv-note: Deidentified free-text clinical notes (version 2.2).

Alistair E. W. Johnson, Lucas Bulgarelli, Lu Shen, Alvin Gayles, Ayad Shammout, Steven Horng, Tom J. Pollard, Sicheng Hao, Benjamin Moody, Brian Gow, Liwei H. Lehman, Leo A. Celi, and Roger G. Mark. 2023b. Mimic-IV, a freely accessible electronic health record dataset. *Scientific Data*, 10(1).

Alistair E. W. Johnson, Tom J. Pollard, Seth J. Berkowitz, Nathaniel R. Greenbaum, Matthew P. Lungren, Chih-ying Deng, Roger G. Mark, and Steven Horng. 2019. Mimic-CXR, a de-identified publicly available database of chest radiographs with free-text reports. *Scientific Data*, 6(1):317.

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

Sunjun Kweon, Jiyoun Kim, Heeyoung Kwak, Dongchul Cha, Hangyul Yoon, Kwang Hyun Kim, Jeewon Yang, Seunghyun Won, and Edward Choi. 2024. Ehrnoteqa: An LLM Benchmark for Real-World Clinical Practice Using Discharge Summaries. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. 2022. Ehrsql: A Practical Text-to-SQL Benchmark for Electronic Health Records. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Chunyuan Li, Cliff Wong, Sheng Zhang, Naoto Usuyama, Haotian Liu, Jianwei Yang, Tristan Naumann, Hoifung Poon, and Jianfeng Gao. 2023. Llava-Med: Training a Large Language-and-Vision Assistant for Biomedicine in One Day. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Tom J. Pollard, Alistair E. W. Johnson, Jesse D. Raffa, Leo A. Celi, Roger G. Mark, and Omar Badawi. 2018. The eICU Collaborative Research Database, a freely available multi-center database for critical care research. *Scientific Data*, 5(1).

Preethi Raghavan, Jennifer J. Liang, Diwakar Mahajan, Rachita Chandra, and Peter Szolovits. 2021. emrkbqa: A Clinical Knowledge-Base Question Answering Dataset. In *Workshop on Biomedical Natural Language Processing (BioNLP)*, pages 64–73.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3980–3990.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, and Remez et al. 2023. Code Llama: Open Foundation Models for Code. *arXiv*.

Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce C. Ho, Carl Yang, and M. D. Wang. 2024. Ehragent: Code Empowers Large Language Models for Few-shot Complex Tabular Reasoning on Electronic Health Records. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, and Bhosale et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv*, abs/2307.09288.

Ping Wang, Tian Shi, and Chandan K. Reddy. 2020. Text-to-SQL Generation for Question Answering on Electronic Medical Records. In *Proceedings of The Web Conference 2020*, pages 350–361. ACM.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, and Yang et al. 2024. Qwen2.5 Technical Report. *arXiv*, abs/2412.15115.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3911–3921.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv*.

Fan Zhou, Siqiao Xue, Danrui Qi, Wenhui Shi, Wang Zhao, Ganglin Wei, Hongyang Zhang, Caigai Jiang, Gangwei Jiang, Zhixuan Chu, and Faqiang Chen. 2024. Db-GPT-Hub: Towards Open Benchmarking Text-to-SQL Empowered by Large Language Models. *arXiv*, abs/2406.11434.

A Appendix

A.1 EHR Dataset Introduction

The **MIMIC-IV (v2.2)** dataset (Johnson et al., 2023b) is a large, publicly accessible relational database containing de-identified health-related data, including diagnoses, procedures, and treatments, for 50,920 patients who were admitted to the critical care units of Beth Israel Deaconess Medical Center (BIDMC) between 2008 and 2019.

The **MIMIC-CXR** dataset (Johnson et al., 2019) is a large-scale, publicly available collection of 377,110 chest radiographs from 227,827 imaging studies conducted at BIDMC between 2011 and 2016. MIMIC-CXR can be linked to MIMIC-IV through lookup tables that map patient identifiers across the two datasets.

The **MIMIC-IV-Note** dataset (Johnson et al., 2023b) is a de-identified collection of free-text clinical notes linked to the MIMIC-IV database. It comprises 331,794 discharge summaries from 145,915 patients (both hospital and emergency department admissions) and 2,321,355 radiology reports from 237,427 patients. All notes have been de-identified in accordance with HIPAA Safe Harbor standards.

We also list the tables and columns used in our dataset in Table 4. There are 18 tables, and all tables are linked by the subject_id and hadm_id. In the MIMIC-CXR dataset, the path of radiology report are stored in the path column, and the discharge summary are stored in the text column in the MIMIMC-Note.

A.2 Dataset Construction

During the dataset construction process, we employed a dual-faceted question design strategy. First, we formulate question templates based on consultations with clinical experts and insights from the relevant literature. Second, we specifically designed complex questions that require the integration of structured data (e.g., tabular information) with unstructured textual data (e.g., radiology reports and discharge summaries). For each standardized question template, we systematically generated multiple paraphrased variants that maintain semantic equivalence, thereby enhancing the diversity and comprehensiveness of the question set. Finally, a sampling approach was used to randomly select one variant from the pool of candidate questions, which was then populated with relevant field values to generate the final question presented to the research subjects. Table 5 lists some question examples related to EHR tables. Table 6 lists some questions related to the text of the CXR reports. Table 7 lists some question examples related to the text of the discharge summaries.

The question answer example is listed in A.4. "question_template" is the template question, "question" is the real question filled with values. "query_code" is the generated query code. "answer" is the answer after running the query code.

Index	Dataset	Table	Columns
1	MIMIC-IV	patients	subject_id, hadm_id, gender, anchor_age, anchor_year, dod.
2		admissions	subject_id, hadm_id, admittime, dischtime, admission_type, admission_location, discharge_location, insurance, marital_status, race.
3		diagnoses	subject_id, hadm_id, icd_code, icd_version.
4		d_icd_diagnoses	icd_code, icd_version, long_title.
5		labevents	subject_id, hadm_id, item_id, charttime, valuenum, valueuom, ref_range_lower, ref_range_upper.
6		d_labitems	itemid, label, fluid, category.
7		microbiolog	subject_id, hadm_id, charttime, spec_type_desc, test_name.
8		prescriptions	subject_id, hadm_id, starttime, stoptime, drug, dose_val_rx, dose_unit_rx, route.
9		procedures	subject_id, hadm_id, icd_code, icd_version.
10		d_icd_procedures	icd_code, icd_version, long_title
11		icustays	subject_id, hadm_id, stay_id, first_careunit, last_careunit, intime, outtime, los.
12		inputevents	subject_id, hadm_id, stay_id, starttime, itemid, amount, amountuom,patientweight, etc.
13		d_items	itemid, label, abbreviation, category, unitname.
14		outputevents	subject_id, hadm_id, stay_id, charttime, itemid, value, valueuom.
15		chartevents	subject_id, hadm_id, stay_id, charttime, itemid, value, valueuom.
16	MIMIC-CXR	cxr-metadata	subject_id, tudy_id, dicom_id, studydate, studytime.
17		cxr-record-list	subject_id, study_id, dicom_id, path.
18	MIMIC-IV-Note	discharge	subject_id, hadm_id, charttime, storetime, text.

Table 4: Dataset, tables, and columns used in our dataset construction.

Prompt

Please generate python code to answer the question.
Use pandas package to load .csv or .csv.gz file.
Only generate code for the question.
No explanation and other description.
Use ‘print‘ to output the result.
The final result variable should be named as ‘result‘.
Questions related to discharge summary should be answered based on the summary-text.

A.3 Dataset Statistics

Here we list some statistics information for our constructed dataset in 8. We classify the difficulty level of the questions based on the number of values that need to be filled in when querying. Questions that require no more than three fill-in values are classified as Level 1 questions, and questions that require more than three fill-in values are classified as Level 2 questions. The more fill-in values required, the more complex the question.

A.4 Prompt Detail

For each table, we provide a detailed explanation of the information conveyed by the table and specify the exact file path from which the table can be accessed. Additionally, for each column within the table, we offer a comprehensive definition that clarifies the specific meaning and significance of the data it represents. Textbox A.4 gives an example of the description to admissions table.

When the model encounters a question it cannot answer, we introduce a mechanism in the prompt

Related Table	Question Template
diagnoses	Has patient {subject_id} been diagnosed with {diagnoses_name} during admission {hadm_id}?
admission	List the hospital admission time of patient {subject_id}.
icu_stay	Count the number of ICU visits of patient {subject_id} during admission {hadm_id}.
labevents	Count the number of {labtest_name} patient {subject_id} received during admission {hadm_id}.
labevents	For patient {subject_id} in admission {hadm_id}, what was the highest value of {labtest_name}?
microbiolog	What are the top [n_rank] frequent microbiology tests that patient {subject_id} had in admission {hadm_id}?
prescriptions	What are the top [n_rank] frequently prescribed drugs of {gender_type} patients aged {age_group} in {year}?
labevents	For patient {subject_id} in admission {hadm_id}, was the last {labtest_name} normal?
prescriptions	Has patient {subject_id} have {durg_name} in his/her {ordinal_num} admission?
microbiology	What was the time that patient {subject_id} have {microbiology_name} in his/her {ordinal_num} admission?

Table 5: Question templates examples related to EHR tables.

Related Table	Question Template
cxr_report	List the {findings_name} of the last chest X-ray study for patient {subject_id} during the hospital stay within admission {hadm_id}.
cxr_report	List the {study_date} of patient {subject_id} who had a chest X-ray study during hospital visit indicating {findings_name} within the admission {hadm_id}.
cxr_report	List the {findings_name} of the chest X-ray study {study_id} for patient {subject_id} during the admission {hadm_id}.
cxr_report	Count the number of chest X-ray study of patient {subject_id} during admission {hadm_id}.
cxr_report	Count the number of {gender} patients aged {age_group} who had a chest X-ray study during hospital visit indicating {findings_name} in the {year}.
cxr_report	Has patient {subject_id} been diagnosed with {diagnoses_name} and also had a chest X-ray study indicating {findings_name} within the admission {hadm_id}?
cxr_report	Has patient {subject_id} received a {procedure_name} procedure and also had a chest X-ray study indicating {findings_name} in the ${natomical_area} within the admission {hadm_id}?
cxr_report	Has patient {subject_id} been prescribed with {drug_name} and also had a chest X-ray study indicating {findings_name} in the ${anatomical_area} within the admission {hadm_id}?

Table 6: Question templates examples related to CXR report text.

Related Table	Question Template
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, does the patient {subject_id} have any known drug allergies?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, what was the patient {subject_id} primary reason for admission?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, what was the patient {subject_id} discharge diagnosis?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, what medications were prescribed to the patient {subject_id} upon discharge?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, what is the family history of the patient {subject_id}?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, describe the hospital course briefly.
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, what medication on admission is given to the patient?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, what was the discharge disposition of the patient?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, was the patient's condition improving?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, list all the blood test items the patient have taken.
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, what happened to the labtest {blood_test_item}?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, why the {blood_test_item} change?
discharge	According to the {ordinal_num} discharge summary of patient {subject_id}, did the patient receive labtest {blood_test_item}?

Table 7: Question templates examples related to discharge summary text.

	Train		Valid		Test	
	Level I	Level II	Level I	Level II	Level I	Level II
Table	2000	2000	500	500	500	500
CXR report	1000	1000	250	250	250	250
Discharge	1000	1000	250	250	250	250
Total	4000	4000	1000	1000	1000	1000

Table 8: Statistics of our dataset

design to ensure a predefined response. Specifically, the model is instructed to return a default value, such as “Unable to answer this question,” when it cannot generate a valid query. This fallback approach improves robustness by providing consistent feedback, even when the question does not match the database schema. By integrating this method, the system remains reliable and predictable, particularly for edge cases or unanswerable queries.

Table Description

This is the description to the admissions.csv.gz file. This file is located in mimic-iv/admissions.csv.gz.
subject_id: A unique identifier for each patient in the dataset. Each patient only has one subject_id.
hadm_id: Hospital admission ID, a unique identifier for each hospital admission. This ID enables differentiation between multiple admissions for the same patient.
admittime: Timestamp for the exact date and time when the patient was admitted to the hospital. This helps establish the start of a hospital stay.
dischtime: Timestamp for the date and time when the patient was discharged from the hospital, marking the end of a specific admission period.
admission_type: Categorical field indicating the type of admission, such as "emergency," "urgent," or "elective." This provides context on the reason or urgency of admission.
admission_location: Describes the location from which the patient was admitted, such as "clinic referral," "emergency department," or "transfer from another facility."
.....

Question Answer Example

"subject_id": 10054277,
"hadm_id": 27607912,
"question_answer_pairs": [
{"question_template": "Count the admission num of patient {subject_id}.",
"question": "How many times does the record show regarding patient 10054277’s admissions?",
"query_code": df = pd.read_csv (’patients.csv.gz’)
result = df[df[’subject_id’] == 10054277][’hadm_id’].nunique()
print(result)
"answer": "1" }

"subject_id": 10054277,
"hadm_id": 27607912,
"question_answer_pairs": [
{"question_template": "What diseases does the patient subject_id have in the admission admission_id according to the radiology report?",
"question": "According to the radiology report, what diseases are associated with patient 10054277 in admission 27607912?",
"query_code": df = pd.read_csv (’patients.csv.gz’)
patient_data = df[(df[’subject_id’] == 10054277) & (df[’hadm_id’] == 27607912)]
report_path = patient_data["path"][0]
question_text = "What disease in the CXR report?"
result = text_func(report_path, question_text) "answer": "atelectasis, pleural effusion." }