# SCARE: A Benchmark for SQL Correction and Question Answerability Classification for Reliable EHR Question Answering

**Gyubok Lee**[*†] GYUBOK.LEE@KAIST.AC.KR
**Woosog Chay**[*†] BENCHAY@KAIST.AC.KR
**Edward Choi**[†] EDWARDCHOI@KAIST.AC.KR
[†] *Korea Advanced Institute of Science & Technology, South Korea*
[*] *Co-first authors*

## Abstract

Recent advances in Large Language Models (LLMs) have enabled the development of text-to-SQL models that allow clinicians to query structured data stored in Electronic Health Records (EHRs) using natural language. However, deploying these models for EHR question answering (QA) systems in safety-critical clinical environments remains challenging: incorrect SQL queries—whether caused by model errors or problematic user inputs—can undermine clinical decision-making and jeopardize patient care. While prior work has mainly focused on improving SQL generation accuracy or filtering questions before execution, there is a lack of a unified benchmark for evaluating independent post-hoc verification mechanisms (i.e., a component that inspects and validates the generated SQL before execution), which is crucial for safe deployment. To fill this gap, we introduce SCARE, a benchmark for evaluating methods that function as a post-hoc safety layer in EHR QA systems. SCARE evaluates the joint task of (1) classifying question answerability (*i.e.*, determining whether a question is answerable, ambiguous, or unanswerable) and (2) verifying or correcting candidate SQL queries. The benchmark comprises 4,200 triples of questions, candidate SQL queries, and expected model outputs, grounded in the MIMIC-III, MIMIC-IV, and eICU databases. It covers a diverse set of questions and corresponding candidate SQL queries generated by seven different text-to-SQL models, ensuring a realistic and challenging evaluation. Using SCARE, we benchmark a range of approaches—from two-stage methods to agentic frameworks. Our experiments reveal a critical trade-off between question classification and SQL error correction, highlighting key challenges and outlining directions for future research.

**Keywords:** Text-to-SQL, EHR QA, Database QA, Reliable QA, Benchmarks and Datasets

**Data and Code Availability** Data and code are publicly available on our GitHub repository at https://github.com/glee4810/SCARE.

**Institutional Review Board (IRB)** IRB approval is not required for this work. The patient records used in this work are from the PhysioNet website and licensed under the Open Data Commons Open Database License v1.0[1].

## 1. Introduction

Electronic Health Records (EHRs) store a wide range of patient data, such as hospital admissions, diagnoses, procedures, and prescriptions, making them essential for healthcare practice and research. Advances in Large Language Models (LLMs) have enabled natural language interaction with EHRs, with text-to-SQL models converting clinicians' questions into SQL queries to retrieve patient data without requiring SQL expertise (Wang et al., 2020; Lee et al., 2022; Bardhan et al., 2024).

However, deploying these systems in clinical settings carries significant risks. For example, an incorrect SQL query could miss a patient's penicillin allergy or miscalculate a medication dosage, leading to potentially catastrophic outcomes. To ensure safe deployment, question answering (QA) systems over EHRs must generate accurate queries that reflect user

[*] These authors contributed equally

1. https://opendatacommons.org/licenses/odbl/1-0/

intent or reject unsuitable ones. However, achieving such reliability is hindered by two key challenges: (1) Clinicians, often unfamiliar with SQL or database systems, may pose problematic questions (Lee et al., 2022). These include *unanswerable* queries (*e.g.*, requesting physician data not in the schema) and *ambiguous* ones (*e.g.*, "BP?") that must be clarified rather than translated directly into SQL. (2) Even well-posed (*answerable*) questions can result in incorrect SQL due to limitations in text-to-SQL models (Tarbell et al., 2023; Lee et al., 2024b; Shi et al., 2024).

To address these challenges, we introduce SCARE[2], a benchmark designed to evaluate a post-hoc reliability layer in EHR QA systems. Unlike existing benchmarks that focus solely on either SQL correction (Pourreza and Rafiei, 2024; Wang et al., 2023b; Askari et al., 2025) or question answerability classification (Zhang et al., 2020; Wang et al., 2023a), SCARE is the first to evaluate an integrated post-hoc layer that handles both tasks, ensuring more reliable QA for EHRs. To construct SCARE, we first source QA data compatible with three major EHR databases—MIMIC-III (Johnson et al., 2016), MIMIC-IV (Johnson et al., 2023), and eICU (Pollard et al., 2018)—and augment it with manually annotated ambiguous and unanswerable questions to capture a diverse range of problematic user queries. We then generate SQL queries from these answerable, ambiguous, and unanswerable questions using a diverse set of text-to-SQL models, from lightweight options to advanced agentic frameworks, and pair them with the corresponding answers (expected model outputs), yielding 4,200 question–SQL–answer tuples. Using SCARE as a testbed, we compare various types of methods. Our experiments reveal significant challenges faced by existing models in handling the integrated task, particularly in balancing the need to correct flawed queries while preserving already-correct ones, and in accurately identifying nuanced ambiguities. These findings highlight the difficulty of robust post-hoc verification and underscore the necessity of SCARE for driving future research toward the safe deployment of clinical QA systems.

2. A benchmark for **S**QL **C**orrection and Question **A**nswerability Classification for **R**eliable **E**HR question answering

## 2. Related Work

### 2.1. EHR QA for Structured Data

EHR question answering (QA) involves answering clinically relevant questions by querying patient data stored in EHRs. These tasks may require clinical knowledge or time-based reasoning. MIMICSQL (Wang et al., 2020) and EHRSQL (Lee et al., 2022), for example, employ text-to-SQL modeling to support question answering over structured EHR databases, demonstrating the potential for querying large-scale datasets using natural language. Similarly, Raghavan et al. (2021) uses a method that first converts natural language questions into logical forms and then translates them into SQL. Alternative approaches have used SPARQL (Park et al., 2021) or Python-based agents (Shi et al., 2024) to perform EHR QA. However, text-to-SQL methods remain the most widely used technology due to their efficiency and scalability for large-scale databases like EHRs.

### 2.2. Reliability in Text-to-SQL

Alongside advances in text-to-SQL models—from few-shot prompting (Rajkumar et al., 2022; Chang and Fosler-Lussier) to advanced task decomposition (Pourreza and Rafiei, 2024) and multi-agent frameworks (Wang et al., 2023b)—a stream of research has focused on enabling more reliable deployment of these models, which can be broadly categorized into three areas.

**SQL Error Correction.** This line of work enhances reliability by reducing model errors in SQL generation, aiming to improve overall performance. The task typically involves taking an initial SQL query as input and producing a corrected version as output (Gong et al., 2025; Askari et al., 2025). Although vital for boosting accuracy, these approaches assume that all input questions are convertible to SQL (*i.e.*, answerable), limiting their applicability in real-world clinical scenarios where user inputs are unconstrained.

**Question Answerability Classification.** A second strand of research focuses exclusively on identifying problematic user questions, typically before SQL generation. Benchmarks like TriageSQL (Zhang et al., 2020) and DTE (Wang et al., 2023a) focus on classifying questions as answerable or one of several non-answerable categories. Although this tackles an essential layer of input validation, these methods act

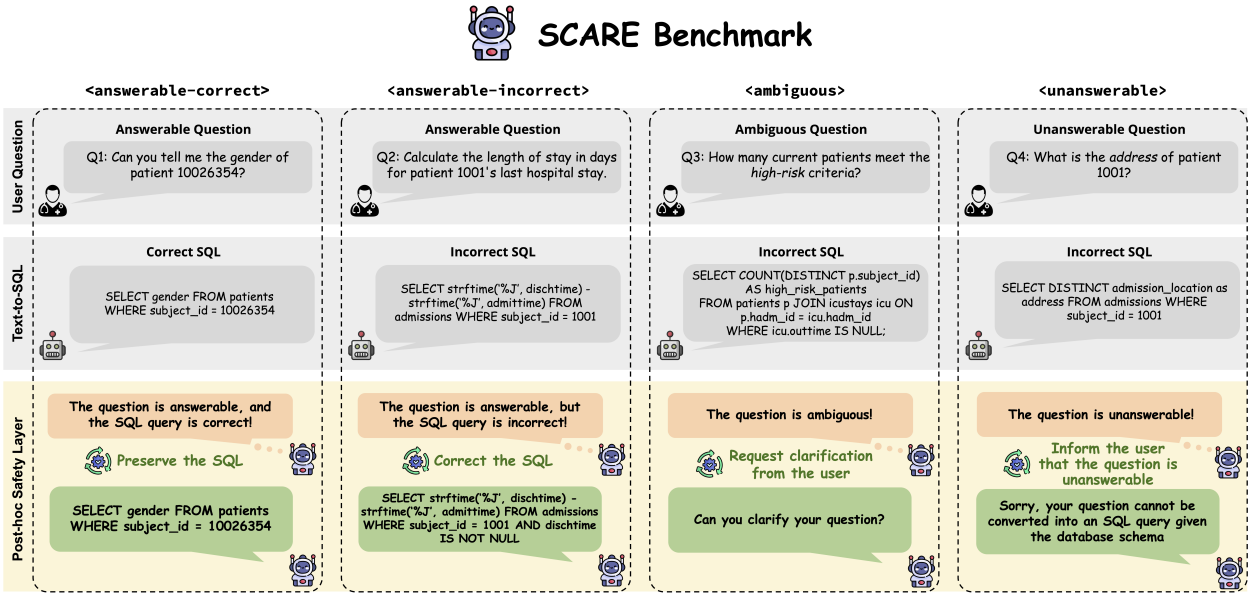


Figure 1: Overview of the SCARE benchmark for evaluating a post-hoc verification layer. The task assumes a candidate SQL query has already been generated by an upstream (and potentially black-box) text-to-SQL model. The layer then takes both the user's question and the candidate SQL as input to decide one of four actions: (1) for an answerable question with correct SQL, preserve the query; (2) for an answerable question with incorrect SQL, correct the query; (3) for an ambiguous question, identify the ambiguity for user clarification; and (4) for an unanswerable question, reject the query and inform the user, thus ensuring the overall reliability and safety of the EHR QA system.

as pre-hoc filters and do not handle SQL correction, resulting in a gray area where neither task fully covers the combined challenge of question filtering and query verification.

**SQL Generation with Abstention.** Most closely related to our goal are works that integrate reliability into the end-to-end generation process. Lee et al. (2022, 2024b,a); Somov and Tutubalina (2025) evaluate models on their ability to generate accurate SQL while abstaining if the query is deemed incorrect, where the concept of incorrect SQL covers both model errors for answerable questions and intrinsic failure due to invalid user input such as ambiguous or unanswerable questions. While these approaches bridge the gap by incorporating caution directly into the pipeline, their evaluation focuses on an all-or-nothing outcome: produce a perfect query or abstain. This conflates the SQL generator's capabilities with the system's verification mechanism, making it difficult to assess their interplay. Furthermore, this all-or-nothing approach limits nuanced user interactions, such as noting that a question is ambiguous or that a request is impossible to fulfill using SQL given the database schema.

## 3. Problem Formulation in SCARE

We define the task of *post-hoc verification*, performed by an independent safety layer within a safety-critical EHR QA system. This layer audits the output of an upstream text-to-SQL model—treated as a black box for modularity—before execution. The task requires three inputs: a natural language question $q$, the database schema $S$, and a candidate SQL query $\hat{y}$. Crucially, $\hat{y}$ is provided for all question types to reflect a real-world deployment scenario where an upstream model might still produce SQL for flawed inputs. The layer must therefore decide whether to preserve the candidate, correct it, or reject it by informing the user.

The SCARE benchmark contains four categories aligned with Fig. 1:

- `answerable-correct` $(q_{ans}, \hat{y}^{cor}, y^*)$: $q_{ans}$ is an answerable question and $\hat{y}^{cor}$ is the correct can-

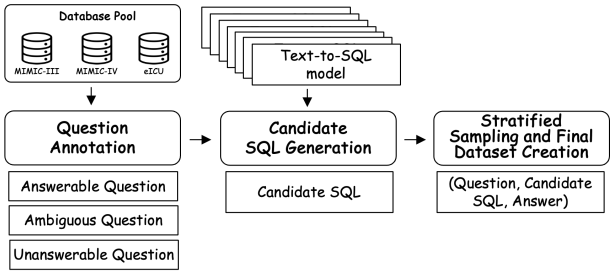


Figure 2: Overview of the SCARE benchmark construction pipeline.

didate SQL. The correctness is determined by comparing its execution result to that of the ground-truth SQL, $y^*$.

- `answerable-incorrect` $(q_{ans}, \hat{y}^{inc}, y^*)$ : The candidate SQL query is incorrect.
- `ambiguous` $(q_{amb}, \hat{y}^{inc}, l_{amb})$: $q_{amb}$ is an ambiguous question and the ground-truth label $l_{amb}$ is the string "ambiguous".
- `unanswerable` $(q_{una}, \hat{y}^{inc}, l_{una})$ : $q_{una}$ is an unanswerable question and the ground-truth label $l_{una}$ is the string "unanswerable".

For both `ambiguous` and `unanswerable`, any candidate $\hat{y}$ is considered incorrect because these questions either require clarification for accurate SQL translation or lie outside the SQL functionalities given the provided schema.

Given $(q, S, \hat{y})$, the model $f$ is expected to output a verified result $o^* \in \{y', \ell_{\text{amb}}, \ell_{\text{una}}\}$:

$$f(q, S, \hat{y}) = \begin{cases} y' & \text{if } q \text{ is answerable} \\ l_{amb} & \text{if } q \text{ is ambiguous} \\ l_{una} & \text{if } q \text{ is unanswerable,} \end{cases} \quad (1)$$

where $y'$ is the output SQL (either the preserved $\hat{y}$ or a corrected SQL query) such that its execution result matches that of the ground-truth SQL $y^*$.

## 4. Benchmark Construction

This section describes the construction process of the SCARE benchmark in three main stages: (1) annotating a diverse set of questions, including answerable, ambiguous, and unanswerable ones, across three EHR databases; (2) generating candidate SQL queries using multiple text-to-SQL models; and (3) conducting stratified sampling to create a balanced evaluation set. An overview of the process is shown in Figure 2.

### 4.1. Question Annotation

**EHR Databases.** Our work uses three major publicly available EHR databases: MIMIC-III (Johnson et al., 2016), MIMIC-IV (Johnson et al., 2023), and eICU (Pollard et al., 2018). For MIMIC-III, we adopt the schema from the MIMICSQL (Wang et al., 2020) dataset. For MIMIC-IV and eICU, we follow the preprocessing procedure used in EHRSQL (Lee et al., 2022). The resulting schema sizes are: MIMIC-III (5 tables, 50 columns), MIMIC-IV (7 tables, 112 columns), and eICU (10 tables, 72 columns).[3]. Using these databases as the foundation for our benchmark, we create a pool of answerable questions, followed by unanswerable and ambiguous ones.

#### 4.1.1. Answerable Question Creation

For questions compatible with MIMIC-III, we use 1,000 question–SQL pairs from MIMICSQL (`mimicsql_natural_v2`). For MIMIC-IV and eICU, however, we cannot directly reuse the EHRSQL pairs, because their value-shuffled patient data do not match our databases. Instead, we construct new data from scratch using the EHRSQL question templates (*e.g.*, "What is the route of administration of {drug_name}?" where the actual value for {drug_name} is later sampled from the database). The construction proceeds in three steps: (1) sample valid values from the databases, (2) generate new ground-truth SQL queries, and (3) paraphrase the questions using OpenAI's GPT-4o. This process results in 450 pairs for MIMIC-IV and 432 pairs for eICU. These SQL queries are considered the ground-truth answers for the answerable portion of our dataset (`answerable-correct` and `answerable-incorrect`).

#### 4.1.2. Ambiguous and Unanswerable Question Annotation

During the deployment of EHR QA systems, users often pose unanswerable or ambiguous questions for which no corresponding SQL exists. To address this

3. We use the demo versions of MIMIC-IV and eICU to avoid privacy alterations such as value shuffling, which perturbs patient data distributions. The demos share the same database schema structures as the full versions.

Table 1: Sample data from the SCARE benchmark. For `answerable-correct`, the model is expected to output the same SQL, as it is correct. For `answerable-incorrect`, the candidate SQL is flawed because it fails to use `DISTINCT` when counting patients, so the model is expected to correct the error in the SQL. For `ambiguous`, the model is expected to output "ambiguous," as the question contains the vague word "enough." For `unanswerable`, the model is expected to output "unanswerable," as "family visitation" goes beyond the information stored in the schema.

| Type | User Question | Candidate SQL | Answer |
|---|---|---|---|
| `answerable-correct` | Count the ICU visits of patient 10007058 since 2100. | `SELECT COUNT(*) FROM icustays WHERE subject_id = 10007058 AND intime >= '2100-01-01 00:00:00' AND intime <= '2100-12-31 23:59:00'` | `SELECT COUNT(*) FROM icustays WHERE subject_id = 10007058 AND intime >= '2100-01-01 00:00:00' AND intime <= '2100-12-31 23:59:00'` |
| `answerable-incorrect` | How many people were admitted to the hospital? | `SELECT count(subject_id) FROM admissions` | `SELECT count(DISTINCT subject_id) FROM admissions` |
| `ambiguous` | How many patients were administered divalproex (delayed release) in *enough* doses since 2100? | `SELECT count(DISTINCT subject_id) FROM prescriptions WHERE drug = 'divalproex' AND stoptime > '2100-01-01'` | `'ambiguous'` |
| `unanswerable` | Did patient 10007795 have *family visitation* during their first ICU stay? | `SELECT ce.charttime AS visitation_time FROM chartevents ce JOIN d_items di ON ce.itemid = di.itemid JOIN icustays icu ...` | `'unanswerable'` |

issue, we curate such questions by annotating new instances into six categories: three types of ambiguity and three types of unanswerability, derived from prior text-to-SQL literature and from frequently unanswerable cases included in EHRSQL Lee et al. (2022). To ensure a diverse and comprehensive set of problematic questions, newly annotated questions are first generated using GPT-4o based on their target categories, followed by human validation to ensure proper alignment between each question and its assigned category, while filtering out semantically similar queries. Descriptions of these categories are provided below, with further annotation details and examples in Appendix E.

**Ambiguous Questions.** Ambiguous questions are those that require clarification before SQL generation (the model answer to these questions is "ambiguous"). These include `vague-question` (**VQ**) instances, such as short, phrasal questions (*e.g.*, "Patient status?") (Radhakrishnan et al., 2020; Wang et al., 2023a); `vague-word` (**VW**) instances with imprecise terms (*e.g.*, "Recent high blood pressure cases") (He et al., 2024); and `ambiguous-reference` (**AR**) instances involving unclear entities (*e.g.*, "The patient from last week") (Yu et al., 2019; Zhu et al., 2024). For reliable EHR QA systems, it is crucial to classify such questions as "ambiguous" and notify users that further clarification is needed to ensure accurate query processing.

**Unanswerable Questions.** Unanswerable questions cannot be resolved via SQL queries given the provided database schema (the model answer to these questions is "unanswerable"). These include `small-talk` (**ST**), encompassing casual talk unrelated to EHR (*e.g.*, "What's the weather like today?") (Zhang et al., 2020); `out-of-scope` (**OS**), where requests go beyond SQL capabilities (*e.g.*, "Predict future patient outcomes") (Zhang et al., 2020; Wang et al., 2023a); and `missing-column`, including newly created examples (**MC′**) and those adapted from EHRSQL (**MC″**), where questions reference non-existent columns (*e.g.*, "Query the blood_type column," but it doesn't exist) (Zhang et al., 2020; Wang et al., 2023a; Lee et al., 2022). For reliable EHR QA systems, it is crucial to classify such questions as "unanswerable" and inform users that the request cannot be fulfilled within the scope of the text-to-SQL task. This notification is important for guiding users to refine their next input to align with the system's capabilities.

Table 2: The dataset statistics of the SCARE benchmark.

| Database | answerable-correct | answerable-incorrect | ambiguous | unanswerable | Total |
|---|---|---|---|---|---|
| **MIMIC-III** | 350 | 350 | 350 | 350 | 1,400 |
| **MIMIC-IV** | 350 | 350 | 350 | 350 | 1,400 |
| **eICU** | 350 | 350 | 350 | 350 | 1,400 |
| **Total** | 1,050 | 1,050 | 1,050 | 1,050 | 4,200 |

### 4.2. Candidate SQL Generation

Building on the questions, we generate candidate SQL queries to simulate real-world model outputs that could be implemented within the EHR QA system.

#### 4.2.1. Generating Candidate SQL

To generate a diverse pool of candidate SQL queries ($\hat{y}$), we utilize a variety of text-to-SQL models, ranging from fine-tuned SQL-specialized models to agentic frameworks. This diverse model set enables stress-testing of error-correction frameworks against varied $\hat{y}$ given $q$ distributions, accommodating diverse question types such as answerable, ambiguous, and unanswerable queries for EHR QA. The models include: **LLM-SQL**, a few-shot baseline utilizing GPT-4o for SQL generation (Chang and Fosler-Lussier); **CodeS-15B**, a 15B parameter model pretrained for SQL and fine-tuned on the BIRD dataset (Li et al., 2024); **DIN-SQL**, an advanced in-context learning approach using GPT-4o with task decomposition and self-correction (Pourreza and Rafiei, 2024); **MAC-SQL**, a multi-agent framework employing GPT-4o for iterative SQL query refinement (Wang et al., 2023b); **Deepseek R1-70B**, a 70B parameter general-purpose reasoning model (Guo et al., 2025); **o4-mini**, an advanced reasoning model from OpenAI; and **Qwen3-32B**, a 32B parameter general-purpose reasoning model (Yang et al., 2025). Table 7 reports the execution accuracy of these models.

### 4.3. Stratified Sampling and Final Dataset Creation

Based on the pool of generated SQL candidates, we construct the final benchmark through stratified sampling, dividing the data into subgroups (strata) to ensure a balanced distribution across different scenarios. For each of the three databases, we select 1,400 instances, evenly balanced across four strata (350 instances each): `answerable-correct`, where the candidate SQL returns the correct answer, matching the ground-truth result; `answerable-incorrect`, where the candidate SQL returns an incorrect answer, often due to issues like wrong column selection or invalid operations; `ambiguous`, where instances are derived from ambiguous questions, making any candidate SQL inherently invalid; and `unanswerable`, where instances are derived from unanswerable questions, making any candidate SQL inherently invalid. After sampling from each database independently, we combine these selections across the three databases to form the complete benchmark, resulting in a well-balanced dataset of 4,200 instances overall (3 databases $\times$ 1,400 instances each). Rather than reflecting a naturally skewed real-world distribution of user queries, this stratified, balanced split ensures that each of the four key safety scenarios across different EHR databases is equally tested, which is essential for the diagnostic evaluation of post-hoc verification models in EHR QA systems. Sample data and data statistics are shown in Tables 1 and 2.

## 5. Experiments

### 5.1. Metrics

For the two `answerable` categories, we use three metrics. First, we measure **Coverage (Cov)**, the proportion of instances where the model provides a SQL output rather than a classification label (*i.e.*, "ambiguous", or "unanswerable"). Then, we measure the **Preservation Rate (PR)** on `answerable-correct` inputs and the **Correction Rate (CR)** on `answerable-incorrect` inputs. Both metrics are defined as the final execution accuracy, calculated over the total number of instances in their respective categories. This ensures that models are penalized not only for errors in SQL generation but also for incorrectly classifying an answerable question as `ambiguous` or `unanswerable`.

For the `unanswerable` and `ambiguous` categories, we evaluate the model's ability to correctly classify questions into their respective categories. We report the per-class **Precision**, **Recall**, and **F1-score** for both the "unanswerable" and "ambiguous" labels.

### 5.2. Methods

To establish comprehensive baselines on SCARE, we evaluate seven methods that represent distinct strategies for the joint task of question answerability classi-

Table 3: Baseline performance on the SCARE benchmark. Methods classify questions as answerable, ambiguous, or unanswerable, and handle SQL queries for answerable questions by preserving correct queries or fixing incorrect ones. Metrics are defined in Section 5.1. Higher values indicate better performance.

| Method | `answerable-correct` | | `answerable-incorrect` | | `ambiguous` | | | `unanswerable` | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **PR** | **Cov** | **CR** | **Cov** | **Prec** | **Rec** | **F1** | **Prec** | **Rec** | **F1** |
| Two-Stage | 80.4 | 81.3 | 42.5 | 77.7 | 72.5 | 36.4 | 48.4 | 58.2 | **93.0** | 71.6 |
| Single-Turn | 97.9 | **99.5** | 49.6 | 97.6 | 84.5 | 31.6 | 46.0 | **84.1** | 70.7 | 76.8 |
| Single-Turn-Veri | **98.1** | **99.5** | 52.1 | **98.1** | 89.0 | 32.4 | 47.5 | 84.0 | 72.5 | 77.8 |
| Multi-Turn-SelfRef | 97.2 | 99.4 | 51.4 | **98.1** | **89.4** | 32.9 | 48.1 | 83.5 | 72.7 | 77.7 |
| Single-Turn-Cls | 95.5 | 97.6 | 53.0 | 94.2 | 87.3 | 39.1 | 54.0 | 75.4 | 86.6 | 80.6 |
| Single-Turn-Veri-Cls | 95.9 | 97.8 | **53.8** | 94.0 | 87.0 | 39.0 | 53.9 | 75.7 | 86.8 | 80.8 |
| Multi-Turn-SelfRef-Cls | 97.8 | 99.4 | 53.0 | 98.1 | 88.0 | 32.9 | 47.9 | 83.3 | 72.7 | 77.6 |

fication and SQL correction. These include four base methods and three hybrid variants built upon them.

- Two-Stage: This method follows a modular, divide-and-conquer strategy. It first employs a dedicated classifier to determine if a question is answerable, ambiguous, or unanswerable. Only if the question is deemed answerable does it proceed to a second stage, where a separate module verifies the candidate SQL and corrects it if necessary.

- Single-Turn: This method adopts an integrated, end-to-end approach. In a single generative pass, the model is tasked with jointly analyzing the question and candidate SQL to simultaneously handle answerability classification and SQL correction, directly outputting either a final SQL query or a classification label.

- Single-Turn-Veri: Building on the end-to-end approach, this method introduces a simple iterative verification loop. It extends Single-Turn by having an internal verifier to check the generated output. If the verifier flags an error, the model performs multiple retries to generate a correct answer.

- Multi-Turn-SelfRef: This method involves an iterative refinement strategy in a multi-turn setting. It generates an initial answer, produces feedback on its own output, and then uses this feedback to guide the next refinement attempt.

The three variant methods—Single-Turn-Cls, Single-Turn-Veri-Cls, and Multi-Turn-SelfRef-Cls—are designed as hybrid approaches. These aim to combine the strengths of the modular and integrated strategies by using the output from the specialized classifier in Two-Stage (i.e., the classification result and its reasoning) as an explicit guiding signal for the integrated models. We provide the detailed prompts used to implement the four base methods in Appendix F.

For the backbone LLMs, we use Gemini-2.5-Flash (Comanici et al., 2025) as the main LLM throughout the main experiments. Additional results for two other closed-source models (GPT-5 mini (OpenAI, 2025) and Gemini-2.0-Flash (Google, 2025)) and two open-source models (Llama-3.3-70B (Meta, 2024) and Qwen3-32B (Yang et al., 2025)) are provided in Appendix A, all of which exhibit similar performance trends when used to implement the methods.

### 5.3. Results

Table 3 shows the performance of various baseline methods on the SCARE benchmark.

**Trade-off between question classification and SQL correction.** A critical challenge lies in balancing the need to preserve correct SQL queries (PR) while accurately identifying ambiguous or unanswerable questions. The Two-Stage approach, which decouples these tasks, achieves the highest recall for `unanswerable` (93.0%). However, this comes at a significant cost to Cov, as the initial classification stage

frequently misidentifies answerable questions. Conversely, Single-Turn excels at PR (97.9%) by integrating the tasks, but its ability to detect ambiguity is notably weak (Recall 31.6%). This suggests an inherent tension where maximizing preservation often leads to overlooking problematic inputs, and vice versa. We provide a qualitative analysis of the error cases in Section B.

**Iterative refinement improves correction, but limitations remain.** Iterative approaches (Single-Turn-Veri and Multi-Turn-SelfRef) demonstrate improvements in CR without compromising PR. Multi-Turn-SelfRef achieves a CR of 51.4%, compared to 49.6% for the basic Single-Turn, while maintaining a high PR (97.2%). However, the overall correction capability is still limited. A detailed breakdown of correction outcomes by SQL error type in Table 4 (with explanations provided in Section D) shows that, although the methods perform reasonably well on localized errors such as Table/-Column (T/C) references (55.1% CR), they struggle considerably with Other Global (OG) errors, which require substantial structural changes to SQL queries (30.8% CR).

**Hybrid approaches yield the best balance.** The most effective strategies leverage the strengths of both decoupled classification and integrated refinement. The -CLS variants, which incorporate the reasoning output from the Two-Stage classifier, significantly enhance overall performance. Notably, Single-Turn-Veri-Cls achieves the highest CR (53.8%) and the best F1 score for `unanswerable` (80.8%), while maintaining a strong PR (95.9%). This hybrid approach effectively mitigates the trade-offs observed in the base methods, pointing towards the necessity of integrating explicit answerability reasoning into the verification process.

**Nuanced ambiguity remains highly challenging.** Methods consistently struggle to identify `ambiguous`, achieving a maximum F1 score of only 54.0%. As detailed in Table 5, detection rates for `vague-question` (VQ) and `vague-word` (VW) are particularly poor, often remaining below 35%. This indicates that while models can easily identify overt issues like `small-talk` (ST, >94% recall), they lack the sensitivity required to detect subtle linguistic ambiguities. This deficiency poses a significant risk, as undetected ambiguities can lead to the execution of incorrect SQL queries.

Table 4: Detailed correction rates by SQL error types for `answerable-incorrect`. T/C, J/G, PV, OL, and OG denote table/column reference errors, JOIN/-GROUP BY errors, predicate value errors, other local errors, and other global errors, respectively.

| CR | T/C | J/G | PV | OL | OG |
|---|---|---|---|---|---|
| Two-Stage | 45.4 | 48.5 | 39.7 | 31.4 | 17.8 |
| Single-Turn | 52.4 | 54.4 | **50.7** | 40.4 | 28.0 |
| Single-Turn-Veri | **55.1** | 56.4 | **50.7** | 41.0 | 29.0 |
| Multi-Turn-SelfRef | **55.1** | **58.8** | **50.7** | **42.0** | **30.8** |

Table 5: Recall for correctly identifying ambiguous and unanswerable questions by granular question categories. See Section 4.1.2 for question type definitions.

| Rec | VQ | VW | AR | ST | OS | MC′ | MC″ |
|---|---|---|---|---|---|---|---|
| Two-Stage | 30.1 | 22.1 | **56.9** | **100.0** | **99.2** | **79.1** | **94.7** |
| Single-Turn | 32.6 | 29.1 | 33.1 | 98.0 | 74.1 | 53.6 | 60.3 |
| Single-Turn-Veri | 33.7 | 29.4 | 34.0 | 98.0 | 77.0 | 54.0 | 64.0 |
| Multi-Turn-SelfRef | 34.8 | 29.9 | 33.7 | 98.0 | 77.0 | 54.8 | 64.0 |
| Single-Turn-Cls | 32.0 | **32.3** | 53.1 | 95.6 | 92.1 | 72.6 | 87.0 |
| Single-Turn-Veri-Cls | 32.6 | 30.5 | 54.0 | 95.6 | 91.2 | 74.1 | 87.0 |
| Multi-Turn-SelfRef-Cls | **37.4** | 31.7 | 52.3 | 94.8 | 91.6 | 74.5 | 87.7 |

### 5.4. Qualitative Analysis of Failure Modes

To better understand the challenges highlighted by our benchmark, further qualitative analysis of model failures is provided in Appendix B. As suggested by quantitative results, the most salient errors fall into two categories: failing to detect nuanced ambiguity and failing to correct global SQL errors.

**Failures in `ambiguous` Classification.** Models consistently fail to identify ambiguity when a question contains vague expressions (e.g., `vague-word`, `vague-question`) and the candidate SQL either ignores or misinterprets them. For example, models often incorrectly approve a candidate SQL query for a question like "Find patients with *sufficient data*" because the query appears syntactically valid, failing to recognize that "sufficient data" is unresolvable without clarification. In other cases, for phrasal questions like "Sodium?", models incorrectly approve a degenerate candidate SQL such as SELECT label FROM d_labitems WHERE label = 'Sodium' instead of classifying the question as ambiguous. These fail-

ures persist even with frontier LLMs. We hypothesize that this is due to the lack of joint consideration of linguistic vagueness handling and SQL generation when building LLMs. As a result, the models are biased towards generating any SQL that seems fit, rather than assessing the semantic answerability of the question itself.

**Failures in `answerable-incorrect` Correction.** For answerable questions, models struggle most with Other Global (OG) errors, which require substantial logical or structural corrections to the candidate SQL. These failures mostly stem from two main issues: limited ability to follow instructions and overreliance on parametric knowledge. First, models violate explicit textual guidelines provided in the prompt. For example, when instructed to use the earliest diagnosis time if multiple records exist, a model often overlooks this instruction and preserves incorrect SQL candidate queries. Second, models hallucinate SQL logic based on their internal knowledge instead of using the database schema provided in the context window. These include cases where models invent non-existent ICD codes or table relationships. These cases show the limitations of current LLMs in strictly following complex instructions and leveraging provided context over internal parametric knowledge.

## 6. Conclusion

The safe deployment of EHR question answering systems in clinical environments demands reliability mechanisms that go beyond standard text-to-SQL generation accuracy. In this work, we introduce SCARE, the first benchmark specifically designed to evaluate a unified post-hoc safety layer tasked with the joint challenge of SQL correction and question answerability classification. Grounded in open-source EHR databases, SCARE incorporates diverse scenarios derived from various text-to-SQL models.

Our comprehensive evaluation reveals critical limitations in current approaches. We uncover a stark trade-off between preserving correct queries and accurately identifying problematic questions (either ambiguous or unanswerable). Furthermore, our experimental results reveal that methods severely struggle to detect nuanced ambiguities commonly posed during EHR QA. While hybrid approaches combining iterative refinement with explicit classification signals show promise, significant advancements are still needed before these systems can be reliably deployed in safety-critical clinical applications. SCARE provides an essential tool for the community to drive future research toward developing robust and auditable verification methods, ultimately facilitating the safe integration of LLMs into clinical workflows.

## 7. Limitations and Future Work

The SCARE benchmark has several limitations rooted in its specific design choices. First, its balanced data distribution is intentionally designed for a diagnostic stress test, ensuring each of the four key scenarios is evenly evaluated. However, this design in turn does not reflect the natural distribution of queries during real-world clinical deployment. Similarly, the benchmark is grounded in academic EHR schemas (MIMIC-III, MIMIC-IV, and eICU), which are smaller than many production systems. This choice was made to foster transparent and reproducible research, though generalization to larger, proprietary schemas remains an important future challenge. Finally, SCARE evaluates user-system interactions in a single-turn setting. This design enables a controlled evaluation of a model's ability to detect various problematic question-candidate SQL pairs without the confounding effects of dialogue history. We recognize that the resolution of ambiguity, as opposed to its mere detection, is often best handled through multi-turn interaction. Future research can extend the SCARE framework to the actual production-level EHR systems and a multi-turn setting.

## Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grants (No.RS-2019-II190075, No.RS-2025-02304967) and National Research Foundation of Korea (NRF) grants (NRF-2020H1D3A2A03100945), funded by the Korea government (MSIT).

## References

Arian Askari, Christian Poelitz, and Xinye Tang. Magic: generating self-correction guideline for in-context text-to-sql. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence*

*and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025. ISBN 978-1-57735-897-8. doi: 10.1609/aaai.v39i22.34511. URL https://doi.org/10.1609/aaai.v39i22.34511.

Jayetri Bardhan, Kirk Roberts, and Daisy Zhe Wang. Question answering for electronic health records: Scoping review of datasets and models. *Journal of Medical Internet Research*, 26:e53636, 2024.

Shuaichen Chang and Eric Fosler-Lussier. How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. In *NeurIPS 2023 Second Table Representation Learning Workshop*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, and et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL https://arxiv.org/abs/2507.06261.

Yue Gong, Chuan Lei, Xiao Qin, Kapil Vaidya, Balakrishnan Narayanaswamy, and Tim Kraska. Sqlens: An end-to-end framework for error detection and correction in text-to-sql, 2025. URL https://arxiv.org/abs/2506.04494.

Google. Gemini 2.0: Flash, flash-lite and pro. https://developers.googleblog.com/en/gemini-2-family-expands/, Feb 2025. Accessed: 2025-09-05.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Xinyi He, Mengyu Zhou, Xinrun Xu, Xiaojun Ma, Rui Ding, Lun Du, Yan Gao, Ran Jia, Xu Chen, Shi Han, et al. Text2analysis: A benchmark of table question answering with advanced data analysis and unclear queries. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18206–18215, 2024.

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. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3 (1):1–9, 2016.

Alistair EW Johnson, Lucas Bulgarelli, Lu Shen, Alvin Gayles, Ayad Shammout, Steven Horng, Tom J Pollard, Sicheng Hao, Benjamin Moody, Brian Gow, et al. Mimic-iv, a freely accessible electronic health record dataset. *Scientific data*, 10(1): 1, 2023.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. Ehrsql: A practical text-to-sql benchmark for electronic health records. *Advances in Neural Information Processing Systems*, 35:15589–15601, 2022.

Gyubok Lee, Woosog Chay, Seonhee Cho, and Edward Choi. Trustsql: Benchmarking text-to-sql reliability with penalty-based scoring. *arXiv preprint arXiv:2403.15879*, 2024a.

Gyubok Lee, Sunjun Kweon, Seongsu Bae, and Edward Choi. Overview of the EHRSQL 2024 shared task on reliable text-to-SQL modeling on electronic health records. In Tristan Naumann, Asma Ben Abacha, Steven Bethard, Kirk Roberts, and Danielle Bitterman, editors, *Proceedings of the 6th Clinical Natural Language Processing Workshop*, pages 644–654, Mexico City, Mexico, June 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.clinicalnlp-1.62. URL https://aclanthology.org/2024.clinicalnlp-1.62.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):1–28, 2024.

Meta. Llama 3.3: Model cards and prompt formats, 2024. URL https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/.

OpenAI. Introducing gpt-5. https://openai.com/index/introducing-gpt-5/, Aug 2025. Accessed: 2025-09-05.

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

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

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36, 2024.

Karthik Radhakrishnan, Arvind Srikantan, and Xi Victoria Lin. Colloql: Robust text-to-sql over search queries. In *Proceedings of the First Workshop on Interactive and Executable Semantic Parsing*, pages 34–45, 2020.

Preethi Raghavan, Jennifer J Liang, Diwakar Mahajan, Rachita Chandra, and Peter Szolovits. emrkbqa: A clinical knowledge-base question answering dataset. Association for Computational Linguistics, 2021.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*, 2022.

Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce Ho, Carl Yang, and May Dongmei Wang. Ehragent: Code empowers large language models for few-shot complex tabular reasoning on electronic health records. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22315–22339, 2024.

Oleg Somov and Elena Tutubalina. Confidence estimation for error detection in text-to-sql systems. *arXiv preprint arXiv:2501.09527*, 2025.

Richard Tarbell, Kim-Kwang Raymond Choo, Glenn Dietrich, and Anthony Rios. Towards understanding the generalization of medical text-to-sql models and datasets, 2023. URL https://arxiv.org/abs/2303.12898.

Bing Wang, Yan Gao, Zhoujun Li, and Jian-Guang Lou. Know what I don't know: Handling ambiguous and unknown questions for text-to-SQL. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5701–5714, Toronto, Canada, July 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.352. URL https://aclanthology.org/2023.findings-acl.352.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*, 2023b.

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

An Yang, Zihan Qiu, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. 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)*, pages 1962–1979, 2019.

Yusen Zhang, Xiangyu Dong, Shuaichen Chang, Tao Yu, Peng Shi, and Rui Zhang. Did you ask a good question? a cross-domain question intention classification benchmark for text-to-sql. *arXiv preprint arXiv:2010.12634*, 2020.

Xiaohu Zhu, Qian Li, Lizhen Cui, and Yongkang Liu. Large language model enhanced text-to-sql generation: A survey. *arXiv preprint arXiv:2410.06011*, 2024.

## Appendix A. Full Performance Results

Table 6 reports the full results for GPT-5 mini, Gemini-2.0-Flash, Llama-3.3-70B, and Qwen3-32B.

## Appendix B. Qualitative Analysis

Through a manual review of incorrect model outputs, we identified several recurring error patterns that highlight the key challenges posed by our benchmark. The primary failure mode for each category is detailed below:

- `answerable-correct`: The most common type of failure was an attempt to modify an already-correct candidate SQL query, which resulted in an incorrect version. It was infrequent for the model to misclassify these questions as not answerable.
- `answerable-incorrect`: The most common failure was the inability to correct the provided incorrect SQL query.
- `ambiguous`: the most common failure was classifying the question as answerable and generating a SQL query without realizing the question's ambiguity. For example, when given the question is "Albumin?" and a candidate SQL provided an empty result, a model may decide to fix the query to not have an empty result rather than classifying the question as ambiguous. This could result in a final output like "SELECT DISTINCT label FROM d_labitems WHERE label = 'albumin"'. Misclassifying an ambiguous question as "unanswerable" was infrequent.
- `unanswerable`: Misclassifying the question as "ambiguous" was as common as misclassifying it as answerable and attempting to provide a SQL query. This was mainly due to models not taking the database schema (even though it was given as an input) into consideration when determining answerability. For example, for the question "What was the name of the diagnosis for patient 10039997 in other departments?", where no columns regarding departments exist, a model might misclassify the question as ambiguous because of the phrase "other departments." This indicates the model failed to incorporate the provided knowledge of the database schema when classifying the question.

## Appendix C. Performance of Text-to-SQL Models for Candidate SQL Generation

Table 7 presents the performance comparison of seven text-to-SQL models across three medical datasets: MIMIC-IV, eICU, and MIMIC-III (MIMICSQL). Notably, OpenAI's o4-mini achieves the highest accuracy, with Qwen3-32B ranking second, while CodeS-15B yields the lowest performance.

## Appendix D. SQL Error Types and Definitions

Table 8 presents the examples of error types present in the candidate SQL query from the answerable questions of the sCARE benchmark. In the table, T/C, J/G, PV, OL, and OG denote table/column reference errors, JOIN/GROUP BY errors, predicate value errors, other local errors, and other global errors, respectively.

## Appendix E. Details of Ambiguous and Unanswerable Question Annotation

We provide further details on the generation and annotation process for the six categories of ambiguous and unanswerable questions introduced in Section 4.1.2. Table 9 shows samples for each category alongside an answerable example.

### E.1. Ambiguous Questions

The following details the generation process for the three types of ambiguous questions.

- `vague-question`: We prompt GPT-4o to create overly vague, keyword-based questions (Radhakrishnan et al., 2020) conditioned on a hospital domain, simulating user queries that lack specific intent (*e.g.*, "Weight?", "Symptoms?").
- `vague-word`: To generate questions with imprecise filtering conditions, we prompt GPT-4o to strategically insert ambiguous terms (e.g., "common," "long," "more than usual") into otherwise answerable questions.

Table 6: Full baseline performance on the SCARE benchmark. Higher values indicate better performance.

| Method | answerable-correct | | answerable-incorrect | | ambiguous | | | unanswerable | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **PR** | **Cov** | **CR** | **Cov** | **Prec** | **Rec** | **F1** | **Prec** | **Rec** | **F1** |
| *GPT-5 mini* | | | | | | | | | | |
| Two-Stage | 82.4 | 85.6 | 55.1 | 83.9 | 68.1 | 73.7 | 70.8 | 77.7 | 88.4 | 82.7 |
| Single-Turn | 95.7 | 99.3 | 62.6 | 98.4 | 75.5 | 44.3 | 55.8 | 87.7 | 64.6 | 74.4 |
| Single-Turn-Veri | 96.0 | 99.4 | 65.0 | 99.1 | 80.8 | 46.0 | 58.6 | 85.5 | 70.7 | 77.4 |
| Multi-Turn-SelfRef | 95.7 | 99.5 | 67.8 | 98.7 | 81.8 | 46.1 | 59.0 | 83.5 | 72.3 | 77.5 |
| Single-Turn-Cls | 88.4 | 91.7 | 59.0 | 86.6 | 74.6 | 75.8 | 75.2 | 79.0 | 88.0 | 83.2 |
| Single-Turn-Veri-Cls | 88.8 | 92.7 | 63.7 | 90.1 | 77.5 | 75.5 | 76.5 | 80.2 | 88.1 | 84.0 |
| Multi-Turn-SelfRef-Cls | 93.2 | 97.9 | 68.1 | 95.9 | 84.9 | 68.5 | 75.8 | 79.6 | 89.0 | 84.0 |
| *Gemini-2.0-Flash* | | | | | | | | | | |
| Two-Stage | 78.6 | 80.0 | 27.5 | 75.5 | 77.3 | 35.6 | 48.8 | 54.3 | 95.7 | 69.3 |
| Single-Turn | 98.8 | 99.5 | 29.4 | 94.7 | 69.2 | 13.2 | 22.2 | 78.5 | 68.5 | 73.1 |
| Single-Turn-Veri | 98.9 | 99.4 | 30.7 | 94.2 | 68.6 | 15.0 | 24.6 | 78.7 | 69.8 | 74.0 |
| Multi-Turn-SelfRef | 97.7 | 98.8 | 34.2 | 91.5 | 68.8 | 18.9 | 29.6 | 74.1 | 76.3 | 75.2 |
| Single-Turn-Cls | 96.3 | 97.6 | 30.1 | 89.9 | 84.9 | 33.2 | 47.8 | 70.6 | 92.6 | 80.1 |
| Single-Turn-Veri-Cls | 96.3 | 97.7 | 31.1 | 89.1 | 85.8 | 34.0 | 48.7 | 70.4 | 92.9 | 80.1 |
| Multi-Turn-SelfRef-Cls | 95.0 | 96.4 | 34.0 | 87.1 | 84.8 | 35.5 | 50.1 | 67.7 | 93.2 | 78.4 |
| *Llama-3.3-70B* | | | | | | | | | | |
| Two-Stage | 69.0 | 73.1 | 29.9 | 68.1 | 71.4 | 47.6 | 57.1 | 52.9 | 99.0 | 68.9 |
| Single-Turn | 60.4 | 98.5 | 20.8 | 90.4 | 48.8 | 18.6 | 26.9 | 75.6 | 69.6 | 72.5 |
| Single-Turn-Veri | 97.1 | 99.2 | 38.1 | 92.3 | 57.6 | 25.3 | 35.2 | 74.5 | 76.5 | 75.5 |
| Multi-Turn-SelfRef | 95.1 | 97.4 | 38.9 | 88.1 | 60.0 | 31.8 | 41.6 | 72.0 | 77.6 | 74.7 |
| Single-Turn-Cls | 74.1 | 77.0 | 29.5 | 69.6 | 74.9 | 49.4 | 59.6 | 54.1 | 99.1 | 70.0 |
| Single-Turn-Veri-Cls | 75.0 | 77.6 | 31.1 | 69.0 | 74.0 | 49.3 | 59.2 | 54.3 | 98.9 | 70.1 |
| Multi-Turn-SelfRef-Cls | 72.7 | 75.3 | 33.0 | 67.3 | 72.1 | 49.1 | 58.4 | 53.6 | 99.0 | 69.5 |
| *Qwen3-32B* | | | | | | | | | | |
| Two-Stage | 73.6 | 75.0 | 22.6 | 71.7 | 60.4 | 51.4 | 55.6 | 58.9 | 94.1 | 72.5 |
| Single-Turn | 51.0 | 93.4 | 18.5 | 84.7 | 60.4 | 25.9 | 36.3 | 71.1 | 68.6 | 69.8 |
| Single-Turn-Veri | 97.8 | 99.2 | 31.8 | 88.5 | 65.9 | 31.2 | 42.4 | 71.5 | 74.0 | 72.7 |
| Multi-Turn-SelfRef | 91.0 | 95.0 | 40.9 | 89.7 | 69.5 | 40.7 | 51.3 | 68.4 | 79.3 | 73.5 |
| Single-Turn-Cls | 78.5 | 81.0 | 27.9 | 74.1 | 60.9 | 50.0 | 54.9 | 61.4 | 94.2 | 74.4 |
| Single-Turn-Veri-Cls | 78.8 | 81.0 | 29.9 | 73.3 | 61.5 | 51.7 | 56.2 | 61.6 | 93.8 | 74.3 |
| Multi-Turn-SelfRef-Cls | 86.1 | 89.9 | 41.0 | 83.9 | 71.7 | 51.5 | 59.9 | 65.8 | 92.0 | 76.7 |

Table 7: Execution accuracy of seven different text-to-SQL models on MIMIC-IV, eICU, and MIMIC-III (MIMICSQL).

| | MIMIC-IV | eICU | MIMICSQL |
|---|---|---|---|
| LLM-SQL | 61.1 | 60.2 | 74.4 |
| CodeS-15B | 24.0 | 15.1 | 62.0 |
| DIN-SQL | 59.8 | 56.3 | 76.9 |
| MAC-SQL | 66.0 | 59.5 | 75.7 |
| Deepseek R1-70B | 46.7 | 54.2 | 76.0 |
| Qwen3-32B | 69.3 | 62.3 | **86.9** |
| OpenAI o4-mini | **72.4** | **69.0** | 85.9 |
| On Average | 40.1 | 38.2 | 65.8 |

- `ambiguous-reference`: We create questions with unresolved references by prompting GPT-4o to modify answerable questions, incorporating referentially ambiguous words like "this," "that," or "them."

### E.2. Unanswerable Questions

We create three types of unanswerable questions to test a system's ability to recognize queries beyond its scope.

- `small-talk`: We use GPT-4o to generate conversational questions unrelated to EHR data (e.g., "Did you grab coffee before rounds today?"), with instructions to explicitly avoid referencing the database schema.
- `out-of-scope`: We generate these by prompting GPT-4o to transform existing answerable questions into analytical tasks that extend beyond SQL's capabilities, such as predictive modeling.
- `missing-column`: In addition to using examples from EHRSQL, we generate a new set of more difficult questions for this category. We increase the difficulty by designing questions that reference columns that are plausible within the EHR context but do not exist in the database schema.

**Quality Check** To ensure the reliability and consistency of our annotations, all generated questions undergo a rigorous review process. The process is conducted by three annotators (two authors and one hired external annotator), all of whom are computer science graduate students proficient in SQL. Each annotator independently evaluates whether the questions fit their designated categories. We measure inter-annotator agreement using Cohen's kappa, which ranges from 85.8% to 90.9%, and Fleiss' kappa, which is 87.8%, indicating a high level of agreement. Only questions that receive unanimous approval from all three annotators are included in the final dataset.

## Appendix F. Baseline Method Implementation

We present the prompts used to implement our four base methods: Two-Stage, Single-Turn, Single-Turn-Veri, and Multi-Turn-SelfRef. Note that the {evidence} part in the prompt refers to assumptions made during SQL annotation that are not explicitly stated in the questions (e.g., use SQLite for SQL query generation; use DENSE_RANK() only when ranking is explicitly specified).

### F.1. Two-Stage

**Prompt 1: Prompt used for the classification stage in Two-Stage.**

```
Your task is to classify a natural language
question into one of the following three
categories, based on whether it can be
answered using SQL over the given database
schema.

Classification Categories:
1. **answerable** - The question can be
clearly answered with the given database
schema. All required information (tables,
columns, relationships) exists in the schema.

2. **ambiguous** - The question is unclear,
ambiguous, or requires clarification. This
includes:
   - Questions with unclear references (it,
   that, the previous one)
   - Questions with multiple possible
   interpretations without further
   clarification
   - Questions that are too vague to
   understand the intent

3. **unanswerable** - The question cannot be
answered with the given database schema. This
 includes:
   - Questions requiring information not
   available in the database
   - Questions that are completely out of
   scope for SQL operations
```

```
   - Questions requiring functionality
   outside of SQL operations
   - Questions that are general conversation
   or small talk

# Input
- Database Schema
- Question

# Output Format
Respond with a single JSON object:
{{
  reasoning: <reasoning behind your decision
  >,
  answer: <one of the three categories:
  answerable, ambiguous, or unanswerable>
}}

# Input
Database Schema:
{database_schema}

Question: {question}
```

**Prompt 2: Prompt used for the SQL correction stage in Two-Stage.**

```
Your task is to check whether the given SQL
query is correct according to the schema, the
 question, and the SQL guideline.
If it is incorrect, provide a corrected query
. If it is correct, return it unchanged.

# Input
- Database Schema
- Question
- Evidence (SQL guideline)
- Current SQL Query
- Execution Result

# Output Format
Respond with a single JSON object:
{{
  reasoning: <reasoning behind your decision
  >,
  answer: the corrected SQL query if
  incorrect, otherwise the original query
}}

# Input
Database Schema:
{database_schema}

Question:
{question}

Evidence:
{evidence}

Current SQL Query:
{sql}

Execution Result:
{exec}
```

### F.2. Single-Turn

**Prompt 3: The prompt used in Single-Turn.**

```
Your task is to determine whether the
predicted SQL is correct, or whether the
question is intrinsically ambiguous or
unanswerable for SQL generation, given the
question and database schema. Follow the
instructions below:

- If the question is **answerable** and the
SQL is **correct**, output the same SQL.
- If the question is **answerable** but the
SQL is **incorrect**, output the fixed SQL.
- If the question is **ambiguous** (requires
clarification before SQL translation), output
 `ambiguous`. Ambiguity types include:
  1. vague-question: Extremely brief or
  unclear questions (e.g., BP?, Patient?).
  2. vague-word: Questions containing vague
  words (e.g., How many patients meet the *
  high risk* criteria?).
  3. referential-ambiguity: Questions
  containing unclear referents (e.g., When
  was the first time *it* happened in the
  most recent hospital visit?).
- If the question is **unanswerable** (cannot
 be converted into valid SQL), output `
unanswerable`. Unanswerable types include:
  1. small-talk: Casual queries unrelated to
  the data (e.g., Did you grab coffee?).
  2. out-of-scope: Requests that cannot be
  handled by SQL (e.g., Can you *cluster*
  patients based on medication patterns?).
  3. missing-column: References to non-
  existent columns (e.g., asking for an *
  address* field not present in the schema).
```

```
# Important
- Do not assume anything that is not
explicitly stated in the input.

# Input
- Database Schema
- Question
- Evidence (SQL guideline)
- Predicted SQL
- Execution Result

# Output Format
Respond with a single JSON object:
{{
  reasoning: <reasoning behind your decision
  >,
  answer: <either the original SQL, fixed SQL
  , ambiguous, or unanswerable>
}}

# Input
Database Schema:
{database_schema}

Question:
{question}

Evidence:
{evidence}

Predicted SQL:
{sql}

Execution Result:
{exec}
```

### F.3. Single-Turn-Veri

**Prompt 4: The SQL correction prompt used in Single-Turn-Veri.**

```
Your task is to determine whether the
predicted SQL is correct, or whether the
question is intrinsically ambiguous or
unanswerable for SQL generation, given the
question and database schema. Follow the
instructions below:

- If the question is **answerable** and the
SQL is **correct**, output the same SQL.
- If the question is **answerable** but the
SQL is **incorrect**, output the fixed SQL.
- If the question is **ambiguous** (requires
clarification before SQL translation), output
 `ambiguous`. Ambiguity types include:
  1. vague-question: Extremely brief or
  unclear questions (e.g., BP?, Patient?).
  2. vague-word: Questions containing vague
  words (e.g., How many patients meet the *
  high risk* criteria?).
  3. referential-ambiguity: Questions
  containing unclear referents (e.g., When
  was the first time *it* happened in the
  most recent hospital visit?).
- If the question is **unanswerable** (cannot
 be converted into valid SQL), output `
unanswerable`. Unanswerable types include:
  1. small-talk: Casual queries unrelated to
  the data (e.g., Did you grab coffee?).
  2. out-of-scope: Requests that cannot be
  handled by SQL (e.g., Can you *cluster*
  patients based on medication patterns?).
  3. missing-column: References to non-
  existent columns (e.g., asking for an *
  address* field not present in the schema).

# Important
- Do not assume anything that is not
explicitly stated in the input.

# Input
- Database Schema
- Question
- Evidence (SQL guideline)
- Predicted SQL
- Execution Result

# Output Format
Respond with a single JSON object:
{{
  reasoning: <reasoning behind your decision
  >,
  answer: <either the original SQL, fixed SQL
  , ambiguous, or unanswerable>
}}

# Input
Database Schema:
{database_schema}

Question:
{question}

Evidence:
{evidence}
```

```
Predicted SQL:
{sql}

Execution Result:
{exec}
```

**Prompt 5: The verifier prompt used in Single-Turn-Veri.**

```
Your task is to verify whether the model has
correctly followed the task instructions for
SQL prediction. Carefully evaluate the
predicted SQL in relation to the database
schema, the question, the evidence (SQL
guideline), and the execution result.

Follow the instructions below:
- If the predicted SQL is correct (or if the
question is not answerable and the label is
ambiguous or unanswerable), start your
feedback with the phrase the predicted SQL is
 correct.
- If the predicted SQL is incorrect, start
your feedback with the phrase the predicted
SQL is incorrect. Then explain clearly why it
 is wrong (e.g., incorrect column, wrong join
, missing condition, misclassification of
ambiguity/unanswerability, etc.).

Ambiguity Types
  1. vague-question: Extremely brief or
  unclear questions (e.g., BP?, Patient?).
  2. vague-word: Questions containing vague
  words (e.g., How many patients meet the *
  high risk* criteria?).
  3. referential-ambiguity: Questions
  containing unclear referents (e.g., When
  was the first time *it* happened in the
  most recent hospital visit?).

Unanswerable Types
  1. small-talk: Casual queries unrelated to
  the data (e.g., Did you grab coffee?).
  2. out-of-scope: Requests that cannot be
  handled by SQL (e.g., Can you *cluster*
  patients based on medication patterns?).
  3. missing-column: References to non-
  existent columns (e.g., asking for an *
  address* field not present in the schema).

# Important
- Your role is to provide evaluation feedback
 only, not to generate or fix SQL
- Your feedback should be precise and
grounded in the given schema and instruction.
- Do not assume anything that is not
explicitly stated in the input.

# Input
- Database Schema
- Question
- Question Explanation
- Evidence (SQL guideline)
- Predicted SQL
- SQL Explanation
- Execution Result

# Output Format
Respond with a single JSON object:
{{
  feedback: <detailed feedback on your
  decision>
}}

# Input
Database Schema:
{database_schema}

Question:
{question}

Evidence:
{evidence}

Predicted SQL:
{sql}

Execution Result:
{exec}
```

### F.4. Multi-Turn-SelfRef

**Prompt 6: Prompt used for the SQL correction stage in Multi-Turn-SelfRef.**

```
Your task is to determine whether the
predicted SQL is correct, or whether the
question is intrinsically ambiguous or
unanswerable for SQL generation, given the
question and database schema. Follow the
instructions below:

- If the question is **answerable** and the
SQL is **correct**, output the same SQL.
```

```
- If the question is **answerable** but the
SQL is **incorrect**, output the fixed SQL.
- If the question is **ambiguous** (requires
clarification before SQL translation), output
 `ambiguous`. Ambiguity types include:
  1. vague-question: Extremely brief or
  unclear questions (e.g., BP?, Patient?).
  2. vague-word: Questions containing vague
  words (e.g., How many patients meet the *
  high risk* criteria?).
  3. referential-ambiguity: Questions
  containing unclear referents (e.g., When
  was the first time *it* happened in the
  most recent hospital visit?).
- If the question is **unanswerable** (cannot
 be converted into valid SQL), output `
unanswerable`. Unanswerable types include:
  1. small-talk: Casual queries unrelated to
  the data (e.g., Did you grab coffee?).
  2. out-of-scope: Requests that cannot be
  handled by SQL (e.g., Can you *cluster*
  patients based on medication patterns?).
  3. missing-column: References to non-
  existent columns (e.g., asking for an *
  address* field not present in the schema).

# Important
- Do not assume anything that is not
explicitly stated in the input.

# Input
- Database Schema
- Question
- Evidence (SQL guideline)
- Predicted SQL
- Execution Result

# Output Format
Respond with a single JSON object:
{{
  reasoning: <reasoning behind your decision
  >,
  answer: <either the original SQL, fixed SQL
  , ambiguous, or unanswerable>
}}

# Input
Database Schema:
{database_schema}

Question:
{question}

Evidence:
{evidence}

Predicted SQL:
{sql}

Execution Result:
{exec}
```

**Prompt 7: Prompt used for the feedback stage in Multi-Turn-SelfRef.**

```
Your task is to review whether the model has
followed the task instructions correctly for
SQL prediction. Carefully examine the
database schema, the question, the evidence (
SQL guideline), the predicted SQL, and the
execution result.

Follow the instructions below:
- If the predicted SQL is correct (or if the
question is not answerable and the label is
ambiguous or unanswerable), start your
feedback with the phrase the predicted SQL is
 correct.
- If the predicted SQL is incorrect, start
your feedback with the phrase the predicted
SQL is incorrect. Then explain clearly why it
 is wrong (e.g., incorrect column, wrong join
, missing condition, misclassification of
ambiguity/unanswerability, etc.).

Ambiguity Types
  1. vague-question: Extremely brief or
  unclear questions (e.g., BP?, Patient?).
  2. vague-word: Questions containing vague
  words (e.g., How many patients meet the *
  high risk* criteria?).
  3. referential-ambiguity: Questions
  containing unclear referents (e.g., When
  was the first time *it* happened in the
  most recent hospital visit?).

Unanswerable Types
  1. small-talk: Casual queries unrelated to
  the data (e.g., Did you grab coffee?).
  2. out-of-scope: Requests that cannot be
  handled by SQL (e.g., Can you *cluster*
  patients based on medication patterns?).
  3. missing-column: References to non-
  existent columns (e.g., asking for an *
  address* field not present in the schema).
```

```
# Important
- Your role is to provide evaluation feedback
 only, not to generate or fix SQL
- Your feedback should be precise, grounded
in the given schema and instruction.
- Do not assume anything that is not
explicitly stated in the input.

# Input
- Database Schema
- Question
- Question Explanation
- Evidence (SQL guideline)
- Predicted SQL
- SQL Explanation
- Execution Result

# Response Format
{{
  feedback: <detailed feedback on your
  decision>
}}

# Input
Database Schema:
{database_schema}

Question:
{question}

Evidence:
{evidence}

Predicted SQL:
{sql}

Execution Result:
{exec}
```

**Prompt 8: Prompt used for the refinement stage in Multi-Turn-SelfRef.**

```
All previous SQL queries were found to be
problematic. Based on the feedback, determine
 once again whether the question is
answerable, ambiguous, or unanswerable. If it
 is answerable, correct the SQL.

Predicted SQL:
{sql}

Execution Result:
{exec}
```

Table 8: Examples of different SQL error types from the answerable questions in the SCARE benchmark. Each row shows a specific error in the candidate query: **T/C**, the query references a wrong column name in the SELECT clause; **J/G**, the query incorrectly joins the wrong table; **PV**, the query uses an incorrect value for the `long_title` column; **OL**, the query unnecessarily uses a ROUND() function; **OG**, the query logic is too simplistic and misses the core user intent.

| Type | User Question | Ground-truth SQL | Candidate SQL |
|---|---|---|---|
| T/C | What was the name of the specimen test that was last given to patient 10027602 since 12/2100? | SELECT microbiologyevents. spec_type_desc FROM microbiologyevents WHERE microbiologyevents.hadm_id IN ( SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = 10027602 ) AND strftime('%Y-%m',microbiologyevents. charttime) >= '2100-12' ORDER BY microbiologyevents.charttime DESC LIMIT 1 | SELECT test_name FROM microbiologyevents WHERE subject_id = 10027602 AND datetime(charttime) >= datetime('2100-12-31 23:59:00','start of month','-0 month') ORDER BY charttime DESC LIMIT 1 |
| J/G | What was patient 10015860's last value of anion gap in 10/this year? | SELECT labevents.valuenum FROM labevents WHERE labevents.hadm_id IN ( SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = 10015860 ) AND labevents.itemid IN ( SELECT d_labitems.itemid FROM d_labitems WHERE d_labitems.label = 'anion gap' ) AND datetime(labevents.charttime,'start of year') = datetime('2100-12-31 23:59:00','start of year','-0 year') AND strftime('%m',labevents.charttime) = '10' ORDER BY labevents.charttime DESC LIMIT 1 | SELECT ce.valuenum FROM chartevents ce JOIN d_items di ON ce.itemid = di.itemid WHERE ce.subject_id = 10015860 AND di.label = 'anion gap' AND datetime(ce.charttime,'start of year') = datetime('2100-12-31 23:59:00','start of year','-0 year') AND strftime('%m', ce.charttime) = '10' ORDER BY ce.charttime DESC LIMIT 1 |
| PV | How many people were given procedure on single vessel two or more times? | SELECT COUNT(T1.subject_id) FROM ( SELECT admissions.subject_id, COUNT(*) AS C1 FROM procedures_icd JOIN admissions ON procedures_icd.hadm_id = admissions.hadm_id WHERE procedures_icd.icd_code = ( SELECT d_icd_procedures.icd_code FROM d_icd_procedures WHERE d_icd_procedures.long_title = 'procedure on single vessel' ) GROUP BY admissions.subject_id ) AS T1 WHERE T1.C1 >= 2 | SELECT COUNT(DISTINCT procedures_icd.subject_id) FROM procedures_icd JOIN d_icd_procedures ON procedures_icd.icd_code = d_icd_procedures.icd_code WHERE d_icd_procedures.long_title = 'single vessel' GROUP BY procedures_icd.subject_id HAVING COUNT(*) >= 2 |
| OL | What was the duration in days of patient 10018845's last stay in the icu? | SELECT strftime('%J', icustays.outtime) - strftime('%J', icustays.intime) FROM icustays WHERE icustays.hadm_id IN ( SELECT admissions.hadm_id FROM admissions WHERE admissions.subject_id = 10018845 ) AND icustays.outtime IS NOT NULL ORDER BY icustays.intime DESC LIMIT 1 | SELECT ROUND((JULIANDAY(outtime) - JULIANDAY(intime)), 0) AS duration_days FROM icustays WHERE subject_id = 10018845 ORDER BY intime DESC LIMIT 1 |
| OG | Can you show me the top three most frequent lab tests in 2100? | SELECT d_labitems.label FROM d_labitems WHERE d_labitems.itemid IN ( SELECT T1.itemid FROM ( SELECT labevents.itemid, DENSE_RANK() OVER ( ORDER BY COUNT(*) DESC ) AS C1 FROM labevents WHERE strftime('%Y',labevents.charttime) = '2100' GROUP BY labevents.itemid ) AS T1 WHERE T1.C1 <= 3 ) | SELECT T1.label, COUNT(*) as frequency FROM d_labitems T1 INNER JOIN labevents T2 ON T1.itemid = T2.itemid |

Table 9: Examples of diverse question types in SCARE.

| Type | User Question | Reason |
|---|---|---|
| `answerable` | What are the five commonly ordered medications for patients aged 60 or above? | Clear and answerable |
| `vague-question` | *BP?* | Too short, unclear intent |
| `vague-word` | How many current patients meet the *high-risk* criteria? | Term *high-risk* not defined |
| `ambiguous-reference` | When was the first time *it* happened in this hospital visit? | Pronoun *it* is ambiguous |
| `small talk` | *Did you grab coffee?* | Not relevant to EHR data |
| `out-of-scope` | Can you *cluster* patients based on their medication patterns? | Beyond text-to-SQL tasks |
| `missing-column` | What is the *address* of patient 10016742? | *Address* not stored in MIMIC-IV/eICU |