

EMR-AGENT: An LLM Agent Framework for Cohort and Feature Extraction from Heterogeneous EHR Databases

Kwanhyung Lee^{1,2*} Sungsoo Hong^{1,3*} Joonhyung Park² Jeonghyeop Lim¹
Juhwan Choi¹ Donghwee Yoon¹ Eunho Yang^{1,2}

¹AITRICS ²Korea Advanced Institute of Science and Technology (KAIST)

³Department of Digital Health, SAIHST, Sungkyunkwan University, Seoul 06355, Korea

{kwanlee9209, sshong}@aitrics.com

Abstract

Electronic medical record (EMR) databases are widely used for clinical machine learning, but their preprocessing remains difficult to reproduce because cohort definitions, patient-level features, and clinical item-code mappings must be adapted to each database schema. We formulate EMR preprocessing as the task of translating study requests into database-specific cohort/feature extraction SQL and local clinical item-code mappings. Existing preprocessing and harmonization frameworks remain difficult to adapt because they rely on dataset-specific rules or predefined input formats. We introduce **EMR-AGENT**, an LLM-agent framework that reduces reliance on manual rule writing by automating database-specific cohort selection, feature extraction, and clinical code mapping from study requests. EMR-AGENT retrieves schema and documentation context, interacts with live databases through SQL, observes execution results, and refines extraction decisions from error feedback. To enable rigorous evaluation, we develop a benchmarking codebase for three intensive care unit (ICU) databases (MIMIC-III, eICU, and SICdb), covering reference databases and a newer SICdb version released after the backbone’s knowledge cutoff. Across these benchmarks, EMR-AGENT reduces schema-debugging and item-code search effort, enabling reproducible evaluation of preprocessing quality under expert review, while reducing reliance on hand-written database-specific pipelines.

1 Introduction

EMR databases contain diverse patient information, including demographics, vital signs, laboratory results, diagnoses, procedures, clinical notes, and outcomes. Recent advances in machine learning (ML) have enabled clinical prediction models to be built from such data (Horn et al., 2020; Li et al., 2023; Luo et al., 2024; Shukla and Marlin, 2021;

Tipirneni and Reddy, 2022), supporting early warning, resource allocation, and risk stratification (Lee et al., 2023a,b). However, building these models requires EMR preprocessing: converting researcher-specified study criteria into structured tabular or time-series inputs through cohort definition, feature extraction, and clinical concept mapping.

In practice, EMR preprocessing remains largely manual and tightly coupled to database-specific schemas. For each database, researchers and clinical experts must inspect documentation, interpret schema conventions, implement extraction logic, and curate mapping dictionaries, limiting reproducibility, comparability, and reuse across institutions (Hur et al., 2022; Jarrett et al., 2021; McDermott et al., 2021).

First, semantic and structural heterogeneity is common across EMR systems. The same clinical concept may be represented by different identifiers, labels, tables, or storage formats across institutions. For example, in three ICU EMR databases used in this work, heart rate appears as `itemid=211` in MIMIC-III, `HeartRateECG` in SICdb, and a `heartrate` column in eICU. These variations change how data must be queried, standardized, and used as model inputs. Existing harmonization frameworks improve consistency within specific settings, but often rely on dataset-specific rules or predefined input formats, making adaptation to new schemas costly (Oliver et al., 2023; van de Water et al., 2024; Xu et al., 2025; Jarrett et al., 2021; McDermott et al., 2023).

Second, ambiguity persists even within a single EMR database. Clinical concepts such as heart rate can be measured through multiple modalities or recorded under multiple item codes, requiring database-specific code mapping decisions (Oliver et al., 2023). Cohort selection instructions are also often underspecified and can be interpreted differently across studies or research groups. For instance, an instruction such as “include patients

*Equal contribution.

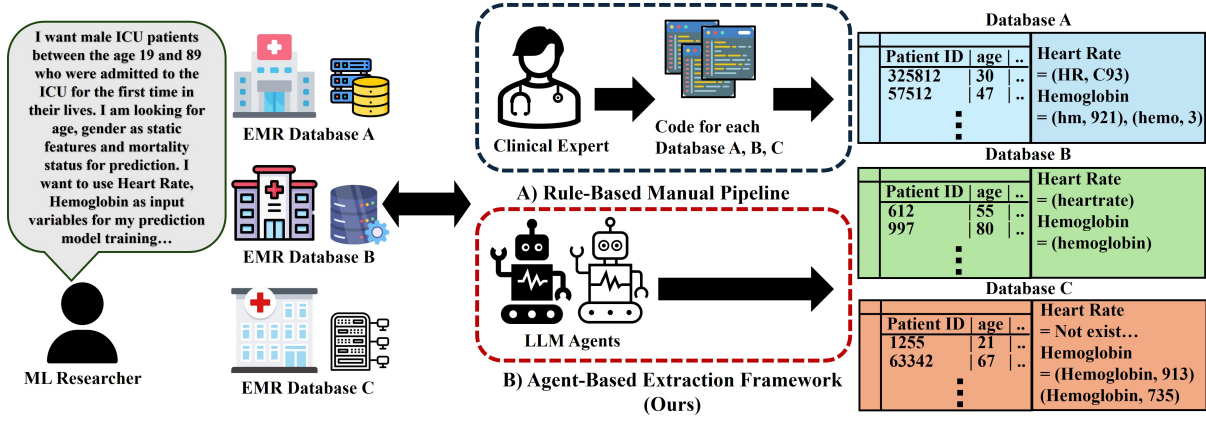


Figure 1: Overview of the shift from (A) the conventional **Rule-Based Manual Pipeline**, where experts handcraft database-specific extraction logic and mapping codes, to (B) **EMR-AGENT**, which assists preprocessing through iterative database interaction across heterogeneous schemas.

admitted to the ICU for the first time” may ambiguously include or exclude patients with previous ICU stays, depending on how admissions, transfers, and repeated ICU stays are defined (Harutyunyan et al., 2019; Purushotham et al., 2018; Wang et al., 2020; Wornow et al., 2023). Thus, even clear clinical intent must be defined through database-specific extraction decisions. This process often requires repeated schema inspection, SQL debugging, and code search, making preprocessing difficult to reproduce and compare across studies.

Recent work on clinical Text-to-SQL and EHR database agents has improved natural-language access to structured clinical data (Lee et al., 2022; Jo et al., 2024; Marshan et al., 2024; Ryu et al., 2024; Shi et al., 2024; Lee et al., 2026; Shen et al., 2026). Yet, EMR preprocessing remains under-explored: unlike database question answering, it requires translating study requests into executable, database-specific preprocessing logic that defines cohorts, constructs patient-level feature tables, and proposes reviewable local item-code candidates.

In this work, we introduce **EMR-AGENT** (Automated Generalized Extraction and Navigation Tool), which, to our knowledge, is the first LLM-agent framework systematically evaluated for EMR preprocessing across cohort selection, feature extraction, and database-specific clinical code mapping. As illustrated in Fig. 1, EMR-AGENT uses modular agents to retrieve schema/document context, query live databases, refine schema assumptions through error feedback, and identify candidate item codes. The resulting SQL and item-code mappings remain inspectable for expert review.

Our contributions are summarized as follows:

- We formulate EMR preprocessing as the task of translating clinical study requests into database-specific cohort definitions, feature extraction logic, and clinical item-code mappings across heterogeneous EMR schemas.
- We propose EMR-AGENT, an LLM-agent framework comprising the Cohort and Feature Selection Agent (CFSA) and the Code Mapping Agent (CMA), to generate database-specific cohort/feature extraction SQL and clinical item-code mappings from study requests.
- We construct benchmarks for MIMIC-III, eICU, and SICdb, spanning reference databases and a newer database version released after the backbone’s knowledge cutoff, and evaluate EMR-AGENT through baselines, ablations, backbone analyses, and a residual-work proxy.

2 Related Work

2.1 EMR Preprocessing and Harmonization

Clinical ML pipelines require EMR preprocessing, including cohort definition, feature extraction, and clinical concept mapping. Prior frameworks improve reproducibility through standardized representations, event streams, or dataset-specific recipes (Wang et al., 2020; Jarrett et al., 2021; Xu et al., 2025; McDermott et al., 2023; van de Water et al., 2024; Oliver et al., 2023), but often rely on predefined formats or hand-written rules that are costly to adapt to new schemas. EMR-AGENT instead translates natural-language preprocessing requests into database-specific extraction logic by reasoning over schemas, documentation, SQL observations, and clinical item codes.

2.2 Clinical Database Querying and Agents

Clinical Text-to-SQL methods and EHR database agents translate natural-language questions into executable SQL queries or code-based database operations (Jo et al., 2024; Marshan et al., 2024; Ryu et al., 2024; Shi et al., 2024), with recent work addressing SQL reliability, question answerability, and cohort reasoning (Lee et al., 2026; Shen et al., 2026). General SQL-agent systems such as CHESS and CHASE-SQL further improve complex SQL synthesis through schema grounding, candidate generation, execution feedback, and iterative refinement (Talaie et al., 2024; Pourreza et al., 2025), while EHRAgent adapts tool-using agents to multi-table EHR question answering (Shi et al., 2024). Unlike these question-answering and SQL-synthesis settings, EMR-AGENT targets the broader preprocessing workflow of constructing database-specific logic for cohort selection, feature extraction, and clinical code mapping across heterogeneous EMR databases.

3 Proposed Framework: EMR-AGENT

EMR-AGENT is an LLM-agent framework for EMR preprocessing over heterogeneous EMR databases. Unlike conventional preprocessing pipelines that encode cohort definitions, feature extraction logic, and item-code mappings as database-specific rules, EMR-AGENT generates database-specific preprocessing outputs from clinical study requests, schema/document context, and SQL execution feedback. Given a user request, the framework produces outputs for two complementary tasks: cohort and feature extraction, and clinical item-code mapping. As shown in Fig. 2, EMR-AGENT consists of two agents: the **Cohort and Feature Selection Agent (CFSA)**, which generates extraction logic for patient cohorts and features, and the **Code Mapping Agent (CMA)**, which identifies database-specific item codes or direct access paths for requested clinical measurements.

Agent Inputs. Both agents use four information sources: (1) a clinical study request specifying the desired cohort, patient-level features, or clinical measurements; (2) schema metadata, including table names, column names, and N sample values per column; (3) database documents, including official manuals and curated notes describing schema semantics and dataset-specific caveats; and (4) a live database interface for SQL execution. The curated notes summarize database-level caveats, such

as excluded tables and identifier conventions, rather than task-specific SQL or gold mappings (Section 4 and Appendix A).

Workflow. EMR-AGENT first links the request to relevant schema and document context and constructs a task-specific Schema Guideline. The guideline explains the roles of linked tables and columns, records supporting schema/document evidence, and identifies missing information needed before finalizing extraction or mapping logic. Each agent then executes SQL queries, inspects intermediate results, and refines its decisions. This structured process mirrors expert EMR preprocessing, where professionals first consult EMR documentation to understand relevant cohorts, features, and local codes, and then explore the EMR database to complete the task with a deeper understanding.

3.1 Schema Linking and Guideline Generation

The CFSA and CMA begin with the **Schema Linking and Guideline Generation** module (Fig. 2a, Fig. 2b). Standard schema linking identifies tables and columns relevant to a clinical study request, whereas EMR preprocessing also requires interpreting how these schema elements should be used to implement the requested cohort, features, or measurements. EMR-AGENT links the request to schema metadata and database documents, such as manuals and curated database notes.

To manage heterogeneous schema and document evidence, we introduce the **Schema Guideline**, a task-specific intermediate representation that records relevant tables and columns, their clinical or preprocessing roles, supporting evidence from documents or sample values, and unresolved ambiguities. Unlike planning-based web agents that use plans to sequence actions (Gu et al., 2024; Gur et al., 2024), the Schema Guideline serves as a diagnostic state: it identifies schema- and value-level information gaps and determines whether SQL-based observation is needed before finalizing extraction or mapping logic.

For CFSA, the Schema Guideline specifies how patient identifiers, admission or ICU-stay tables, demographic fields, outcome fields, and clinical-event tables support cohort and feature extraction. It also flags unresolved encodings, join paths, or value constraints that make final extraction SQL unreliable. For example, in step 1 of Fig. 2a, the guideline identifies the missing encoding of gender,

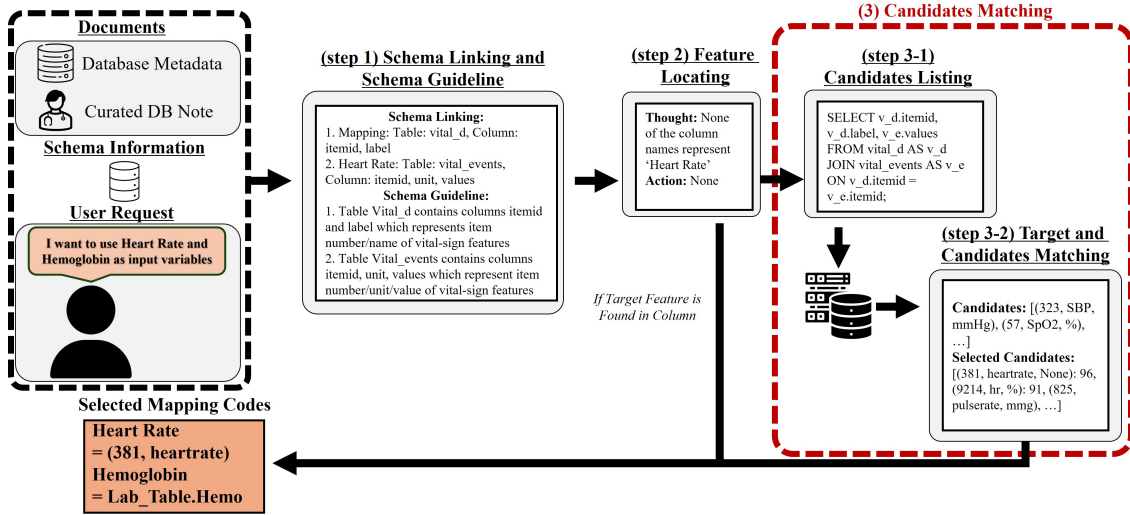
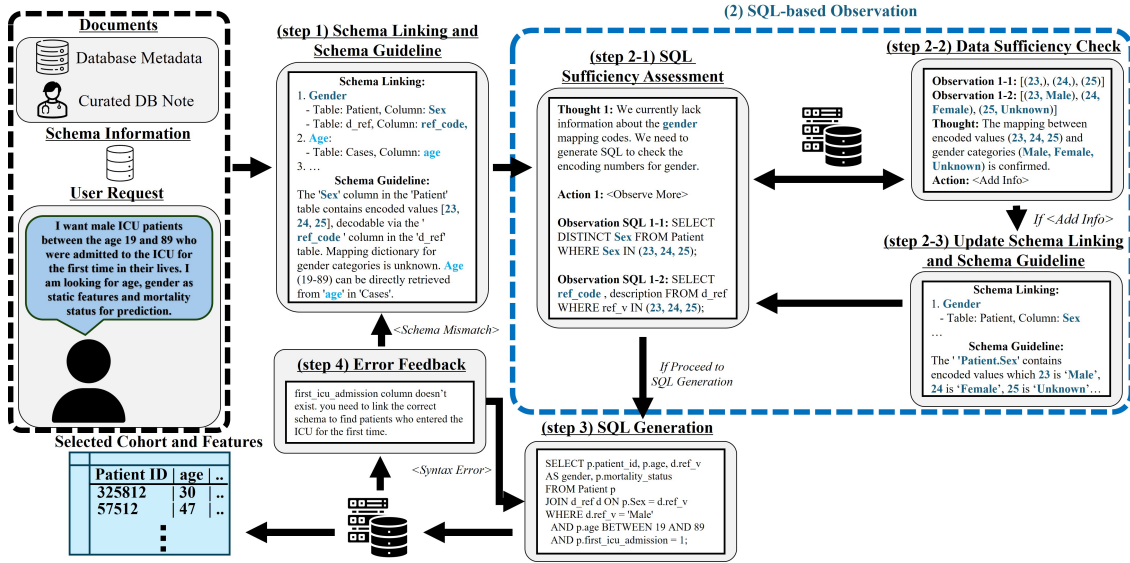


Figure 2: Overview of EMR-AGENT: (a) CFSA performs cohort and feature extraction; (b) CMA performs database-specific clinical code mapping.

indicating that SQL-based observation is needed.

3.2 Cohort and Feature Selection Agent (CFSA)

Beyond Schema Linking and Guideline Generation, CFSA consists of three components: **SQL-based Observation**, **SQL Generation**, and **Error Feedback** (Fig. 2a, Appendix G.1).

SQL-based Observation. This module resolves information gaps identified in the Schema Guideline by interacting with the live EMR database. It iterates over three steps:

- **SQL Sufficiency Assessment:** The agent determines whether the current guideline is sufficient for generating final extraction SQL. If not, it formulates observation SQL queries to inspect

For CMA, the Schema Guideline identifies schema locations that may represent a requested measurement, including direct feature columns, item-code columns, item-label columns, value columns, and unit columns. This lets CMA determine whether to access a measurement directly from a table column or resolve it through candidate item-code listing and matching. For example, in Fig. 2b, Schema Guideline identifies columns representing the item number of vital signs.

missing evidence, such as value encodings, sample values, candidate join keys, or value ranges. For example, in step 2-1 of Fig. 2a, CFSA cannot safely filter male patients because the gender encoding is unknown, and therefore queries the database to identify how gender is represented.

- **Data Sufficiency Check:** After executing the observation SQL queries, the agent checks whether the retrieved results resolve the specific ambiguity recorded in the guideline. Informative observations are passed to Schema Update; otherwise, the agent repeats the sufficiency assessment with a revised observation query. In step 2-2 of Fig. 2a, CFSA observes that the reference code for “Male” is 23, resolving the previously missing encoding.
- **Schema Update:** The agent updates the linked schema and Schema Guideline with newly observed evidence, and uses the updated guideline for subsequent SQL generation.

SQL Generation and Error Feedback. Using the resulting Schema Guideline, CFSA generates final extraction SQL, executes it against the database, and categorizes feedback as follows:

- **Syntactic Error:** Queries with syntax errors are regenerated based on the returned error message.
- **Execution or Schema Error:** Syntactically valid queries that fail at execution time, refer to missing columns, or use incompatible types are sent back to Schema Linking and Guideline Generation with the error message as feedback.
- **Runtime Success:** Queries that execute successfully and return outputs in the expected shape are treated as valid CFSA outputs. They are not assumed to be semantically correct until benchmark evaluation.

This feedback loop is retried up to the maximum number of attempts specified in Section 5, allowing CFSA to revise incorrect schema assumptions and reduce manual SQL debugging.

3.3 Code Mapping Agent (CMA)

CMA maps user-requested clinical measurements to database-specific access paths: either direct table-column locations or local item-code mappings. Similar to CFSA, CMA begins with Schema Linking and Guideline Generation, then applies two modules: **Feature Locating** and **Candidates Matching**(Fig. 2b, Appendix G.2).

Feature Locating. CMA first checks whether the requested measurement is represented directly as a column in the linked schema. If a column is found,

CMA returns the corresponding table and column names as the database-specific access path. If not, the agent treats the measurement as potentially stored as row-level item values, or absent from the database, and proceeds to Candidates Matching.

Candidates Matching. This process begins with listing candidates. CMA identifies tables and columns that may contain measurement identifiers, measurement labels, and unit columns. It then generates SELECT DISTINCT queries to retrieve candidate tuples from these columns, such as item-code, item-label, and unit combinations.

Given the candidate list, CMA performs **Target-Candidate Matching**. The agent compares the requested measurement with candidate tuples in batches, assigns a similarity score from 0 to 100, and retains candidates above a predefined threshold. For example, in step 3-2 of Fig. 2b, CMA compares the requested feature with listed candidates and keeps the candidates above the similarity threshold. The threshold controls the expert-review trade-off: lower values retain more possible mappings for verification, whereas higher values produce smaller candidate lists by filtering potential false positives.

4 EMR Preprocessing Benchmark: *PreCISE-EMR*

Existing benchmarks primarily focus on downstream prediction performance rather than the preprocessing steps that construct cohorts, features, and clinical code mappings. We therefore construct *PreCISE-EMR* (Preprocessing for Cohort Identification, Feature Selection, and Code Extraction), a standardized evaluation protocol and codebase for assessing EMR preprocessing quality, developed with clinical experts.

4.1 Database Environment Setup

We use three publicly available EMR datasets: MIMIC-III (v1.4) (Johnson et al., 2016), eICU (v2.0) (Pollard et al., 2019), and SICdb (v1.0.8) (Rodemund et al., 2024a,b) (Table A.1). These datasets are set up with the official open-source scripts¹², which ensure consistent data processing and loading into PostgreSQL environments. For SICdb, we manually convert the provided CSV files

¹MIMIC-III: <https://github.com/MIT-LCP/mimic-code/tree/main/mimic-iii/buildmimic/postgres>

²eICU: <https://github.com/MIT-LCP/eicu-code/tree/main/build-db/postgres>

into PostgreSQL. The resulting environments generate evaluation sets comparing EMR-AGENT’s outputs with expert-curated reference outputs.

We use MIMIC-III and eICU as reference-database settings and SICdb v1.0.8 as a newer database-version setting released after the backbone’s knowledge cutoff. MIMIC-III and eICU were released before the knowledge cutoff of our primary controlled backbone, Claude-3.5-Sonnet, whereas SICdb v1.0.8 was released after that cutoff. We therefore use SICdb as a post-cutoff database-version setting to evaluate whether EMR-AGENT can adapt to a database version unavailable to the backbone used in our controlled comparisons. Additionally, compared to MIMIC-III (26 tables) and SICdb (7 tables), eICU’s schema is more intricate with 31 tables and features appearing as both column names and row values, making schema parsing and data extraction more challenging.

4.2 Ground-truth Construction

Cohort and feature selection We define evaluation sets focusing on harmonizability and reliability. Harmonizability ensures that our agent consistently selects the same patient groups and features across three heterogeneous databases, enabling the creation of compatible datasets for downstream models. To achieve this, we construct a Cohort and Feature Selection evaluation set by varying exclusion criteria (e.g., age, gender, minimum clinical records, etc.) to generate multiple complex cohorts (Table A.2) on the varying EMR databases. Reliability is assessed by verifying whether our benchmark code, when using the same cohort criteria, selects the same patient groups as existing benchmarks (Harutyunyan et al., 2019; Sheikhalishahi et al., 2020) (Fig. A.1, A.2).

Code mapping Following the approach of curated database notes (Fig. A.3), we select a total of 56 features including vital signs and laboratory results (Table A.3). All features are searched in the Athena Observational Health Data Sciences and Informatics databases (Observational Health Data Sciences and Informatics, 2023) for clinical concepts and are defined using standard terminology. For each dataset, two medical doctors, two nurses, and one healthcare data scientist constructed the mapping dictionary by reviewing database manuals, definition tables, local item labels, units, and standardized terminology. The finalized dictionaries were reviewed by the expert team and serve

as the ground truth for evaluating code mapping (Fig. A.4).

4.3 Evaluation Process

To assess EMR preprocessing quality, we evaluate EMR-AGENT using the Cohort and Feature Selection evaluation sets and the expert-curated code-mapping dictionaries constructed in Section 4.2.

Cohort and feature selection. CFSA is evaluated over 10 repeated trials by comparing predicted ICU-stay IDs with reference ICU-stay IDs and by checking whether the requested patient-level features are returned in the required format. We report F1 for cohort selection and feature-format accuracy for requested outputs such as gender, age, mortality status, and length of stay, capturing both patient selection and output construction quality.

Code mapping. CMA is evaluated over 3 repeated trials by comparing predicted item-code or direct-column access-path tuples with expert-curated mapping dictionaries. We report mapping F1 as the primary metric and absence-aware balanced accuracy (bAcc) to account for concepts absent from a database.

Benchmark access. PreCISE-EMR is a benchmarking framework, not a redistributed dataset. It requires users to obtain appropriate credentials, e.g., via PhysioNet, and execute the code locally; thus, no derived patient-level data are redistributed.

5 Experiments

We evaluate EMR-AGENT along four axes: fixed-backbone comparisons, component/document ablations, residual-work and cost-performance analyses, and open-weight backbone experiments.

Tables 1–3 use Claude-3.5-Sonnet (Anthropic, 2024a) as the fixed backbone for all methods, isolating framework-level effects across baselines and ablations while covering both reference databases and the SICdb post-cutoff database-version setting relative to this controlled backbone. Table 4 evaluates locally deployed Qwen3.5-72B (Qwen Team, 2026) for open-weight transfer and same-backbone adapted-baseline comparison. Appendix B.1 quantifies residual clinician correction and code-search effort. Appendices B.5 and H report cost-performance and data-governance details.

Comparison protocol. For each comparison, all methods use the same backbone, benchmark inputs,

Table 1: Comparison of EMR-AGENT with adapted Text-to-SQL and generic agent baselines under the fixed-backbone setting. MIMIC-III and eICU are reference databases, whereas SICdb v1.0.8 is a newer database version released after the fixed backbone’s knowledge cutoff. CFSA reports F1 and accuracy; CMA reports mapping F1 and absence-aware bAcc.

(a) Cohort and Feature Selection						
Method	MIMIC-III		eICU		SICdb	
	F1	Acc.	F1	Acc.	F1	Acc.
Ours	0.94 $\pm$ 0.01	0.893 $\pm$ 0.01	0.929 $\pm$ 0.03	0.951 $\pm$ 0.03	0.814 $\pm$ 0.04	0.794 $\pm$ 0.04
ICL(PLUQ) (Jo et al., 2024)	0.749 $\pm$ 0.04	0.809 $\pm$ 0.04	0.132 $\pm$ 0.04	0.131 $\pm$ 0.04	0.407 $\pm$ 0.02	0.428 $\pm$ 0.02
ICL(SeqSQL) (Ryu et al., 2024)	0.04 $\pm$ 0.01	0.173 $\pm$ 0.04	0.00 $\pm$ 0.0	0.00 $\pm$ 0.0	0.04 $\pm$ 0.04	0.08 $\pm$ 0.05
DinSQL (Pourreza and Rafiei, 2023)	0.726 $\pm$ 0.05	0.72 $\pm$ 0.04	0.00 $\pm$ 0.0	0.00 $\pm$ 0.0	0.071 $\pm$ 0.03	0.036 $\pm$ 0.02
REACT (Yao et al., 2023)	0.308 $\pm$ 0.05	0.308 $\pm$ 0.04	0.524 $\pm$ 0.06	0.542 $\pm$ 0.06	0.503 $\pm$ 0.04	0.493 $\pm$ 0.03

(b) Code Mapping						
Method	MIMIC-III		eICU		SICdb	
	F1	bAcc.	F1	bAcc.	F1	bAcc.
Ours	0.516 $\pm$ 0.0	0.283 $\pm$ 0.01	0.648 $\pm$ 0.05	0.336 $\pm$ 0.03	0.536 $\pm$ 0.03	0.38 $\pm$ 0.02
ICL(PLUQ) (Jo et al., 2024)	0.022 $\pm$ 0.01	0.036 $\pm$ 0.0	0.125 $\pm$ 0.01	0.112 $\pm$ 0.01	0.119 $\pm$ 0.0	0.078 $\pm$ 0.00
REACT (Yao et al., 2023)	0.214 $\pm$ 0.05	0.14 $\pm$ 0.01	0.067 $\pm$ 0.0	0.081 $\pm$ 0.0	0.218 $\pm$ 0.0	0.154 $\pm$ 0.00

and task-specific LLM-call budget unless otherwise specified. This controls access to database knowledge, so performance differences reflect workflow design rather than privileged hidden rules.

Baselines. Because no prior system directly addresses joint cohort selection, feature extraction, and database-specific code mapping, we adapt representative Text-to-SQL, database-agent, and EHR-agent baselines. For controlled Claude-3.5-Sonnet experiments, we compare against PLUQ-style in-context Text-to-SQL (Jo et al., 2024), SeqSQL (Ryu et al., 2024), DinSQL (Pourreza and Rafiei, 2023), and ReAct (Yao et al., 2023). For the Qwen3.5-27B open-weight comparison in Table 4(b), we adapt CHESS (Talaie et al., 2024), CHASE-SQL (Pourreza et al., 2025), and EHRAgent (Shi et al., 2024) to CFSA and CMA. SeqSQL and DinSQL are evaluated only for CFSA because they target SQL generation rather than item-code matching; implementation details for adapted baselines are provided in Appendix E.

Hyperparameter setting For token efficiency, schema information includes 10 sample values per column. CFSA allows up to 10 observations (5 queries each), with temperature set to 0 for the first 5 observations and increasing by 0.1 thereafter. The Error Feedback module permits 5 retries.

5.1 Comparison with Baselines

Table 1 shows that EMR-AGENT outperforms adapted baselines across heterogeneous EMR schemas. CFSA achieves 0.94, 0.93, and 0.81 F1 on MIMIC-III, eICU, and SICdb, respectively, while the strongest baseline remains below 0.75 and 0.53 on MIMIC-III and eICU, and is approxi-

mately 0.50 on SICdb. CMA improves mapping F1 by 0.30–0.52 over the best competitor, indicating substantially better code-mapping support.

Human-effort impact. Beyond accuracy, Appendix B.1 quantifies residual expert work after automatic generation, with CMA and CFSA details in Tables B.2 and B.1. For CMA, high-cost search operations decrease from 933 to 399, a 57% reduction in the most labor-intensive part of code mapping. For CFSA, clause-level accounting shows a 54–63% reduction in logic errors compared with baselines, decreasing residual correction cases from 339–416 to 155 across 210 outputs. Together, these results show that EMR-AGENT shifts preprocessing from writing and searching from scratch toward verifying compact, inspectable drafts.

5.2 Component Ablation

Table 2 shows that DB Interaction module (SQL-based Observation + Error Feedback) is the most critical component in CFSA, with its removal causing the largest performance drops across all databases. Schema Guideline also yields consistent gains on all datasets. For CMA, Candidates Matching is indispensable, as disabling it collapses performance to near zero, while Schema Guideline further improves robustness across databases.

5.3 Role of External Knowledge

Table 3 shows that documents are important for both CFSA and CMA, especially CMA. Removing documents consistently reduces performance, and removing both documents and modules causes near-complete CMA collapse and large CFSA drops. Nevertheless, EMR-AGENT without documents still outperforms document-equipped baselines in

Table 2: Ablations of (a) CFSA from component removal, (b) CMA from disabling Candidate Matching and SchemaGuideline. DB Interact* represents both SQL-based Observation/Error Feedback.

(a) Cohort and Feature Selection						
Method	MIMIC-III		eICU		SICdb	
	F1	Acc.	F1	Acc.	F1	Acc.
Ours	0.94 $\pm$ 0.01	0.893 $\pm$ 0.01	0.929 $\pm$ 0.03	0.951 $\pm$ 0.03	0.814 $\pm$ 0.04	0.794 $\pm$ 0.04
Ours w/o SQL-based Observation	0.916 $\pm$ 0.01	0.881 $\pm$ 0.01	0.898 $\pm$ 0.03	0.951 $\pm$ 0.03	0.795 $\pm$ 0.05	0.602 $\pm$ 0.04
Ours w/o Error Feedback	0.688 $\pm$ 0.05	0.668 $\pm$ 0.05	0.624 $\pm$ 0.06	0.642 $\pm$ 0.06	0.617 $\pm$ 0.06	0.572 $\pm$ 0.05
Ours w/o DB Interaction*	0.677 $\pm$ 0.05	0.648 $\pm$ 0.05	0.562 $\pm$ 0.06	0.57 $\pm$ 0.06	0.57 $\pm$ 0.06	0.428 $\pm$ 0.05
Ours w/o SchemaGuideline	0.827 $\pm$ 0.03	0.825 $\pm$ 0.01	0.87 $\pm$ 0.03	0.892 $\pm$ 0.04	0.792 $\pm$ 0.05	0.692 $\pm$ 0.04

(b) Code Mapping						
Method	MIMIC-III		eICU		SICdb	
	F1	bAcc.	F1	bAcc.	F1	bAcc.
Ours	0.516 $\pm$ 0.0	0.283 $\pm$ 0.01	0.648 $\pm$ 0.05	0.336 $\pm$ 0.03	0.536 $\pm$ 0.03	0.38 $\pm$ 0.02
Ours w/o Candidates Matching	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.07 $\pm$ 0.0	0.035 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0
Ours w/o SchemaGuideline	0.508 $\pm$ 0.0	0.285 $\pm$ 0.02	0.575 $\pm$ 0.02	0.329 $\pm$ 0.0	0.342 $\pm$ 0.01	0.209 $\pm$ 0.01

Table 3: Ablation results for CFSA and CMA under document removal and module-level ablations.

(a) Cohort and Feature Selection						
Method	MIMIC-III		eICU		SICdb	
	F1	Acc.	F1	Acc.	F1	Acc.
Ours	0.94 $\pm$ 0.01	0.893 $\pm$ 0.01	0.929 $\pm$ 0.03	0.951 $\pm$ 0.03	0.814 $\pm$ 0.04	0.794 $\pm$ 0.04
Ours w/o Documents	0.844 $\pm$ 0.07	0.854 $\pm$ 0.06	0.917 $\pm$ 0.03	0.952 $\pm$ 0.03	0.748 $\pm$ 0.05	0.64 $\pm$ 0.05
Ours w/o Documents, Modules	0.443 $\pm$ 0.05	0.499 $\pm$ 0.05	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.427 $\pm$ 0.06	0.222 $\pm$ 0.03

(b) Code Mapping						
Method	MIMIC-III		eICU		SICdb	
	F1	bAcc.	F1	bAcc.	F1	bAcc.
Ours	0.516 $\pm$ 0.0	0.283 $\pm$ 0.01	0.648 $\pm$ 0.05	0.336 $\pm$ 0.03	0.536 $\pm$ 0.03	0.38 $\pm$ 0.02
Ours w/o Documents	0.336 $\pm$ 0.03	0.19 $\pm$ 0.02	0.322 $\pm$ 0.03	0.208 $\pm$ 0.01	0.138 $\pm$ 0.03	0.072 $\pm$ 0.02
Ours w/o Documents, Modules	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.07 $\pm$ 0.0	0.035 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0

Table 4: Open-weight transfer and adapted-baseline comparison. (a) Backbone transfer to Qwen3.5-27B. (b) Same-backbone comparison with adapted SQL-agent and EHR-agent baselines. Full metrics and LLM-call counts are in Table B.4.

(a) Backbone transfer				
Task	Backbone	MIMIC-III	eICU	SICdb
CFSA	Claude-3.5-Sonnet	0.940(.010)	0.929(.030)	0.814(.040)
	Qwen3.5-27B	0.934(.002)	0.890(.004)	0.992(.000)
CMA	Claude-3.5-Sonnet	0.516(.000)	0.648(.050)	0.536(.030)
	Qwen3.5-27B	0.578(.000)	0.593(.001)	0.603(.000)

(b) Adapted-agent comparison with Qwen3.5-27B				
Task	Method	MIMIC-III	eICU	SICdb
CFSA	EMR-AGENT	0.934(.002)	0.890(.004)	0.992(.000)
	CHES-adapted	0.731(.013)	0.891(.008)	0.397(.015)
	CHASE-SQL-adapted	0.932(.003)	0.902(.008)	0.429(.016)
	EHRAgent-adapted	0.654(.014)	0.357(.015)	0.071(.008)
CMA	EMR-AGENT	0.578(.000)	0.593(.001)	0.603(.000)
	CHES-adapted	0.402(.000)	0.072(.000)	0.538(.000)
	CHASE-SQL-adapted	0.257(.000)	0.500(.000)	0.259(.000)
	EHRAgent-adapted	0.019(.000)	0.238(.000)	0.000(.000)

CFSA and in most CMA settings, suggesting that its task-specific workflow for guideline construction, SQL observation, and candidate matching adds robustness beyond document use alone. Appendix B.2 further shows that SQL-based observation provides fallback schema and value evidence when documentation is unavailable.

5.4 Open-Weight Deployability

The previous experiments use a fixed Claude-3.5-Sonnet backbone to control for model variation.

We next ask two complementary questions. First, does EMR-AGENT transfer to a locally deployable open-weight backbone? Second, under the same backbone, can SQL-agent and EHR-agent adaptations close the gap?

Table 4(a) shows that EMR-AGENT maintains strong CFSA/CMA performance with Qwen3.5-27B, suggesting that the workflow is not tied to a single proprietary API. For SICdb, we interpret this result as open-weight backbone transfer rather than a post-cutoff database-version test for Qwen, since Qwen3.5-27B is newer than the controlled Claude-3.5-Sonnet setting.

Table 4(b) fixes Qwen3.5-27B and compares EMR-AGENT with adapted CHES (Talaei et al., 2024), CHASE-SQL (Pourreza et al., 2025), and EHRAgent (Shi et al., 2024) baselines. Although CHASE-SQL is comparable or slightly stronger in some CFSA reference-database settings, EMR-AGENT achieves the strongest CFSA performance on SICdb and the best CMA F1 across all three databases. Together with the LLM-call analysis in Appendix B.4, these results suggest that EMR preprocessing benefits from task-specific schema guidelines, SQL-based observation, and explicit item-code matching rather than generic SQL-agent or EHR question-answering adaptations alone.

6 Conclusion

We presented EMR-AGENT, an LLM-agent framework for EMR preprocessing across heterogeneous EMR databases. Fixed-backbone comparisons, ablations, residual-work analyses, and open-weight backbone experiments show that EMR-AGENT improves cohort/feature extraction and item-code mapping while remaining suitable for human-in-the-loop review. These results support reproducible evaluation of EMR preprocessing quality and suggest that EMR-AGENT can assist experts without requiring hand-written database-specific pipelines.

Limitations

EMR-AGENT is designed as a human-in-the-loop preprocessing assistant, not as a replacement for clinical experts in EMR preprocessing. Its outputs require expert validation before downstream modeling or clinical use, particularly when resolving ambiguous database-specific item definitions. Our evaluation focuses on ICU databases and common cohort/feature extraction and code-mapping tasks; more complex concepts such as medications, ventilation, and temporally compositional phenotypes require additional reference standards. To support reproducibility, we will release prompts, evaluation scripts, run configurations, and the benchmark codebase.

Ethical Considerations: Data Governance and Model Use

All experiments use de-identified, credentialed-access PhysioNet datasets. PreCISE-EMR is released as a benchmarking codebase rather than a redistributed dataset: users must obtain the required credentials and execute the benchmark locally, and no derived patient-level records are redistributed. The controlled Claude-3.5-Sonnet-20240620 experiments were conducted before August 2025 under the then-public PhysioNet online-service guidance and are reported as fixed-backbone controlled comparisons. Under the updated PhysioNet LLM guidance, our open-weight experiments use locally deployed Qwen3.5-72B; in these runs, database contents, SQL observations, and generated SQL remain within the local execution environment. Appendix H provides the full model-use timeline and release details.

Author Contributions

Kwanhyung Lee conceived, designed, developed, and evaluated EMR-AGENT. Sungsoo Hong designed and constructed the PreCISE-EMR benchmark and its evaluation methodology. Joonhyung Park contributed to manuscript organization and results analysis. Jeonghyeop Lim conducted preliminary studies and implemented baseline models. Juhwan Choi implemented the ReAct baseline. Donghwee Yoon was responsible for the EMR database setup. Eunho Yang supervised the project and provided guidance on the research design and manuscript.

Acknowledgments

This work was supported by National Research Foundation of Korea (NRF) grant (No.RS-2023-00209060, A Study on Optimization and Network Interpretation Method for Large-Scale Machine Learning) funded by the Korea government (MSIT). We would also like to thank Dohee Han, Saebom Lee, and Taeyong Sim as the clinician experts for their valuable support in feature mapping and selection.

References

- Anthropic. 2024a. [Claude 3.5 Sonnet Model Card Addendum](#).
- Anthropic. 2024b. [Model card addendum: Claude 3.5 Haiku and upgraded Claude 3.5 Sonnet](#).
- Anthropic. 2025. [Claude 3.7 Sonnet System Card](#).
- Martin Faltys, Matthias Zimmermann, Xinrui Lyu, Stephanie Hüser, Michael Hyland, Gunnar Rätsch, and Tobias Merz. 2021. HiRID, a high time-resolution ICU dataset (version 1.1.1). <https://doi.org/10.13026/nkwc-js72>. PhysioNet.
- Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. 2000. [PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals](#). *Circulation*, 101(23):e215–e220.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. [The Llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.
- Yu Gu, Boyuan Zheng, Boyu Gou, Kai Zhang, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. 2024. [Is your LLM secretly a world](#)

- model of the internet? model-based planning for web agents. *Preprint*, arXiv:2411.06559.
- Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. [A real-world webagent with planning, long context understanding, and program synthesis](#). In *Proceedings of ICLR*.
- Hrayr Harutyunyan, Hrant Khachatryan, David C Kale, Greg Ver Steeg, and Aram Galstyan. 2019. [Multitask learning and benchmarking with clinical time series data](#). *Scientific data*, 6(1):96.
- Max Horn, Michael Moor, Christian Bock, Bastian Rieck, and Karsten Borgwardt. 2020. [Set functions for time series](#). In *Proceedings of ICML*, pages 4353–4363. PMLR.
- Kyunghoon Hur, Jiyoung Lee, Jungwoo Oh, Wesley Price, Younghak Kim, and Edward Choi. 2022. [Unifying heterogeneous electronic health records systems via text-based code embedding](#). In *Proceedings of CHIL*, pages 183–203. PMLR.
- Daniel Jarrett, Jinsung Yoon, Ioana Bica, Zhaozhi Qian, Ari Ercole, and Mihaela van der Schaar. 2021. [Clairvoyance: A pipeline toolkit for medical time series](#). In *Proceedings of ICLR*.
- Yongrae Jo, Seongyun Lee, Minju Seo, Sung Ju Hwang, and Moontae Lee. 2024. [LG AI Research and KAIST at EHRSQL 2024: Self-training large language models with pseudo-labeled unanswerable questions for a reliable Text-to-SQL system on EHRs](#). In *Proceedings of NAACL 2024 Clinical Natural Language Processing Workshop*, pages 635–643.
- Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. [MIMIC-III, a freely accessible critical care database](#). *Scientific data*, 3(1):1–9.
- Gyubok Lee, Woosog Chay, and Edward Choi. 2026. [SCARE: A benchmark for SQL correction and question answerability classification for reliable EHR question answering](#). In *Proceedings of the Fifth Machine Learning for Health Symposium*, volume 297 of *Proceedings of Machine Learning Research*, pages 1446–1466. PMLR.
- 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 *Proceedings of NeurIPS*, pages 15589–15601.
- Kwanhyung Lee, Soojeong Lee, Sangchul Hahn, Hee-jung Hyun, Edward Choi, Byungeun Ahn, and Joohyung Lee. 2023a. [Learning missing modal electronic health records with unified multi-modal data embedding and modality-aware attention](#). In *Proceedings of MLHC*, pages 423–442. PMLR.
- Kwanhyung Lee, John Won, Hee-jung Hyun, Sangchul Hahn, Edward Choi, and Joohyung Lee. 2023b. [Self-supervised predictive coding with multimodal fusion for patient deterioration prediction in fine-grained time resolution](#). In *Proceedings of ICLR 2023 Workshop on Trustworthy Machine Learning for Healthcare*.
- Zekun Li, Shiyang Li, and Xifeng Yan. 2023. [Time series as images: Vision transformer for irregularly sampled time series](#). In *Proceedings of NeurIPS*, pages 49187–49204.
- Yicheng Luo, Zhen Liu, Linghao Wang, Binqian Wu, Junhao Zheng, and Qianli Ma. 2024. [Knowledge-empowered dynamic graph network for irregularly sampled medical time series](#). In *Proceedings of NeurIPS*, pages 67172–67199.
- Alaa Marshan, Anwar Nais Almutairi, Athina Ioannou, David Bell, Asmat Monaghan, and Mahir Arzoky. 2024. [MedT5SQL: A transformers-based large language model for Text-to-SQL conversion in the healthcare domain](#). *Frontiers in Big Data*, 7:1371680.
- Matthew McDermott, Bret Nestor, Peniel Argaw, and Isaac S Kohane. 2023. [Event stream GPT: A data pre-processing and modeling library for generative, pre-trained transformers over continuous-time sequences of complex events](#). In *Proceedings of NeurIPS (Datasets and Benchmarks Track)*, pages 24322–24334.
- Matthew BA McDermott, Shirly Wang, Nikki Marinsek, Rajesh Ranganath, Luca Foschini, and Marzyeh Ghassemi. 2021. [Reproducibility in machine learning for health research: Still a ways to go](#). *Science Translational Medicine*, 13(586):eabb1655.
- Observational Health Data Sciences and Informatics. 2023. [ATHENA: OHDSI vocabulary repository](#).
- Matthieu Oliver, Jérôme Allyn, Rémi Carencotte, Nicolas Allou, and Cyril Ferdynus. 2023. [Introducing the BlendedICU dataset, the first harmonized, international intensive care dataset](#). *Journal of Biomedical Informatics*, 146:104502.
- Tom Pollard, Alistair Johnson, Jesse Raffa, Leo A. Celi, Omar Badawi, and Roger Mark. 2019. [eICU collaborative research database \(version 2.0\)](#). <https://doi.org/10.13026/C2WM1R>. PhysioNet.
- Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Serkan O. Arik. 2025. [CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in Text-to-SQL](#). In *Proceedings of ICLR*.
- Mohammadreza Pourreza and Davood Rafiei. 2023. [DIN-SQL: Decomposed in-context learning of Text-to-SQL with self-correction](#). In *Proceedings of NeurIPS*, pages 36339–36348.

- Sanjay Purushotham, Chuizheng Meng, Zhengping Che, and Yan Liu. 2018. [Benchmarking deep learning models on large healthcare datasets](#). *Journal of biomedical informatics*, 83:112–134.
- Qwen Team. 2026. [Qwen3.5-72B](#). Hugging Face model card.
- Niklas Rodemund, Andreas Kokoefer, Bernhard Wernly, and Crispiana Cozowicz. 2024a. [Salzburg Intensive Care database \(SICdb\), a freely accessible intensive care database](#). *PhysioNet*. Version 1.0.8.
- Niklas Rodemund, Bernhard Wernly, Christian Jung, Crispiana Cozowicz, and Andreas Koköfer. 2024b. [Harnessing big data in critical care: Exploring a new european dataset](#). *Scientific Data*, 11(320).
- Jaehye Ryu, Seonhee Cho, Gyubok Lee, and Edward Choi. 2024. [EHR-SeqSQL: A sequential Text-to-SQL dataset for interactively exploring electronic health records](#). In *Findings of ACL*, pages 16388–16407.
- Seyedmostafa Sheikhalishahi, Vevake Balaraman, and Venet Osmani. 2020. [Benchmarking machine learning models on multi-centre eICU critical care dataset](#). *Plos one*, 15(7):e0235424.
- Yifei Shen, Yilun Zhao, Justice Ou, Tinglin Huang, and Arman Cohan. 2026. [Patient-similarity cohort reasoning in clinical Text-to-SQL](#). In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1367–1412, Rabat, Morocco. Association for Computational Linguistics.
- Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce C Ho, Carl Yang, and May Dongmei Wang. 2024. [EHRAgent: Code empowers large language models for few-shot complex tabular reasoning on electronic health records](#). In *Proceedings of EMNLP*, pages 22315–22339.
- Satya Narayan Shukla and Benjamin M Marlin. 2021. [Multi-time attention networks for irregularly sampled time series](#). In *Proceedings of ICLR*.
- Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. [CHESS: Contextual harnessing for efficient SQL synthesis](#). *arXiv preprint arXiv:2405.16755*.
- Sindhu Tipirneni and Chandan K Reddy. 2022. [Self-supervised transformer for sparse and irregularly sampled multivariate clinical time-series](#). *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 16(6):1–17.
- Robin van de Water, Hendrik Nils Aurel Schmidt, Paul Elbers, Patrick Thorat, Bert Arnrich, and Patrick Rockenschaub. 2024. [Yet another ICU benchmark: A flexible multi-center framework for clinical ml](#). In *Proceedings of ICLR*.
- Shirly Wang, Matthew McDermott, Geeticka Chauhan, and Marzyeh Ghassemi. 2020. [MIMIC-Extract: A data extraction, preprocessing, and representation pipeline for MIMIC-III](#). In *Proceedings of CHIL*, pages 222–235.
- Michael Wornow, Rahul Thapa, Ethan Steinberg, Jason Fries, and Nigam Shah. 2023. [EHRSHOT: An EHR benchmark for few-shot evaluation of foundation models](#). In *Proceedings of NeurIPS*, pages 67125–67137.
- Justin Xu, Jack Gallifant, Alistair E. W. Johnson, and Matthew B. A. McDermott. 2025. [ACES: Automatic cohort extraction system for event-stream datasets](#). In *Proceedings of the International Conference on Learning Representations*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2025. [Qwen2.5 technical report](#). *Preprint*, arXiv:2412.15115.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [ReAct: Synergizing reasoning and acting in language models](#). In *Proceedings of ICLR*.

A Details of *PreCISE-EMR*: Preprocessing Benchmark

A.1 EMR Database Description

Table A.1 summarizes the EMR databases included in our *PreCISE-EMR* benchmark. We use three publicly available EMR databases, MIMIC-III (v1.4), eICU (v2.0), and SICdb (v1.0.8), so that EMR-AGENT and all baselines can be evaluated under the same reproducible database setup (see Section 4.1).

Our benchmark, *PreCISE-EMR*, provides hard-coded preprocessing code for two evaluation tasks: (1) Cohort and Feature Selection and (2) Code Mapping.

A.2 Cohort and Feature Selection

A.2.1 Benchmark Construction

For the evaluation of cohort and feature selection, we release a hard-coded benchmark that allows users to specify cohort and feature selection variables. The benchmark enables users to control commonly used inclusion and exclusion criteria, including: 1) age, 2) gender, 3) missing discharge information, 4) minimum ICU stay duration, 5) exclusion of patients with multiple ICU stays, 6) missing gender information, and 7) minimum number of clinical records. These criteria are referenced from well-established studies (Harutyunyan et al., 2019; Sheikhalishahi et al., 2020; van de Water et al., 2024; Wornow et al., 2023). To ensure reliability, we validate our benchmark code using the same cohort criteria as prior benchmarks (Harutyunyan et al., 2019; Sheikhalishahi et al., 2020), confirming that our code extracts identical patient lists under identical criteria (see Figs. A.1 and A.2).

A.2.2 Evaluation Set for Cohort and Feature Selection

Using the released benchmark code (Appendix A.2.1), we construct evaluation sets with natural language inputs that specify (a) user-defined *inclusion and exclusion criteria* (for cohort selection) and (b) user-requested *features* (for feature selection), as summarized in Table A.2 following the column of *Cohort Selection (CS)* and *Feature Selection (FS)*.

³MIMIC-III: <https://github.com/YerevaNN/mimic3-benchmarks/tree/v1.0.0-alpha>

⁴eICU: <https://github.com/mostafaalishahi/eICU-Benchmark>

For each evaluation set, the agent must (i) identify the correct cohort (ICU Stays list), with the corresponding patient list reported as **ICU Stays** for each database, and (ii) extract the requested features for these patients in the requested format from *Feature Selection*. Cohort selection accuracy is evaluated by comparing the predicted ICU stay IDs to the gold-standard IDs using the F1-score. Feature Selection accuracy is measured by the correctness of extracted values for the requested features for the patients ICU stays, as shown in Table A.2.

Note that evaluation sets 5, 6, and 7 include (*CMA output*), indicating that mapping codes are provided. For the cohort and feature selection tasks, ground-truth mapping codes are used, as the performance of the code mapping task is evaluated separately. Each evaluation set was run 10 times (for a total of 70 scores), and the final results were obtained by averaging across trials.

A.3 Code Mapping

A.3.1 Code Mapping Construction

As described in Section 4.2, we collaborated with a team of five clinical experts (see Fig. A.3) to construct code-mapping dictionaries for MIMIC-III, eICU, and SICdb.

We began with 118 standardized observation variables from HiRID v1.1.1. We retained 57 concepts recurring across established ICU time-series feature sets, including the MIMIC-III benchmark, BlendedICU, and YAIB. We additionally included 14 clinically relevant concepts standardized using Athena/OHDSI, resulting in 71 candidate concepts. We then restricted the scope to vital signs and laboratory measurements with sufficiently clear cross-database definitions, yielding the final 56 concepts.

The mapping dictionary was constructed in three stages. First, the clinical data expert compiled per-database candidate mappings for the 71 concepts using database manuals, definition tables, local labels, table context, units, and available metadata. Two medical doctors and two nurses independently reviewed these candidates and flagged ambiguous or questionable mappings. Second, flagged entries were jointly reconciled with the clinical data expert by examining schema structure and table semantics; mappings were retained by consensus or excluded when a reliable database-local mapping could not be established. Third, the reconciled set was restricted to the final vital-sign and laboratory scope. We did not compute formal inter-annotator agree-

Table A.1: Types and purposes of datasets used in this study.

Dataset	Version	Published	Use	Purpose
MIMIC-III (Johnson et al., 2016)	1.4	May 2016	✓	EMR database environment
eICU (Pollard et al., 2019)	2.0	Apr. 2019	✓	EMR database environment
SICdb (Rodemund et al., 2024a,b)	1.0.8	Sep. 2024	✓	EMR database environment
HiRID (Faltys et al., 2021)	1.1.1	Feb. 2021	△	Reference for the feature list

Note. Since feature names in HiRID (v1.1.1) (Faltys et al., 2021) are defined using standard terminology, HiRID was used only as a reference for selecting the mapping-code feature list and was excluded from the EMR database environment. For MIMIC-III and eICU, we used the official source code: MIMIC-III¹ and eICU².

¹<https://github.com/MIT-LCP/mimic-code/tree/main/mimic-iii/buildmimic/postgres>

²<https://github.com/MIT-LCP/eicu-code/tree/main/build-db/postgres>

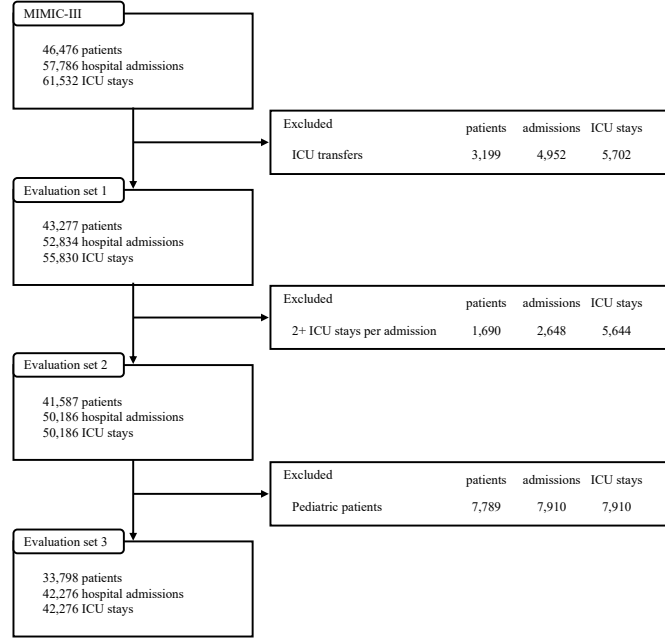


Figure A.1: A flowchart for comparison of MIMIC-III benchmark³ as a reliability evaluation.

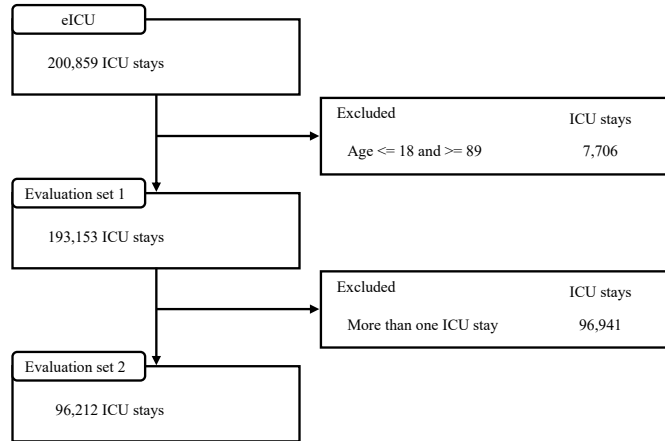


Figure A.2: A flowchart for comparison in eICU benchmark⁴ as a reliability evaluation.

ment because this was an iterative expert-curation and consensus-reconciliation process rather than a blinded annotation study.

A.3.2 Evaluation Set for Code Mapping

Our benchmark *PreCISE-EMR* provides an input set of 56 standardized features, referenced from OHDSI (Observational Health Data Sciences and Informatics, 2023) and listed in Table A.3. Be-

Table A.2: User-requested Inclusion and Exclusion criteria (Cohort Selection) applied for Harmonizability evaluation and User-Requested Feature Format (Feature Selection). The base cohorts corresponding to ICU stays in MIMIC-III, eICU, and SICdb are 61,532, 200,859, and 27,350, respectively. (CMA output) represents the prediction output from (Code Mapping Agent).

Evaluation set	Cohort Selection (CS) and Feature Selection (FS)	ICU Stays (N)		
		MIMICIII	eICU	SICdb
1	CS: Include only Age 19 to 29 and Include only Male and Exclude ICU stays with missing discharge time FS: ICU-stay id, gender (Male/Female/Unknown), age (integer), length of stay (hours, rounded to 4 decimals in float format)	1,303	4,797	428
2	CS: Include only Age 61 to 69 and Include only Female and Include only ICU stays with at least 30 hours duration FS: ICU-stay id, gender (Male/Female/Unknown), age (integer), mortality status (Dead/Alive/Unknown)	2,960	10,257	519
3	CS: Include only Age 70 to 89 and Include only Male and Exclude stay with multiple ICU stays FS: ICU-stay id, gender (Male/Female/Unknown), age (integer), mortality status (Dead/Alive/Unknown)	5,603	18,387	4,965
4	CS: Include only ICU stays from patients aged 20 to 30 and Exclude patient with missing gender information and Include both Female and Male patients FS: ICU-stay id, gender (Male/Female/Unknown), age (integer), mortality status (Dead/Alive/Unknown)	2,326	9,705	1,158
5	CS: Include only ICU stays from patients aged 40 to 55 and include ICU stays which contains at least one clinical record of 'Hemoglobin [Mass/volume] in Arterial blood (CMA output)' FS: ICU-stay id, gender (Male/Female/Unknown), age (integer), mortality status (Dead/Alive/Unknown)	10,748	36,094	4,911
6	CS: Include only ICU stays from patients aged 19 to 30 and Include only Male patients and include stays which contains at least 15 clinical record of 'Bicarbonate [Moles/volume] in Arterial blood(CMA output)' FS: ICU-stay id, gender (Male/Female/Unknown), age (integer), mortality status (Dead/Alive/Unknown)	339	470	206
7	CS: Include only ICU stays from patients aged 55 to 70 and include ICU stays which contains at least one clinical record of 'Lactate [Mass/volume] in Arterial blood(CMA output)' or 'Methemoglobin/Hemoglobin.total in Arterial blood(CMA output)' FS: ICU-stay id, gender (Male/Female/Unknown), age (integer), mortality status (Dead/Alive/Unknown)	10,574	27,915	11,666

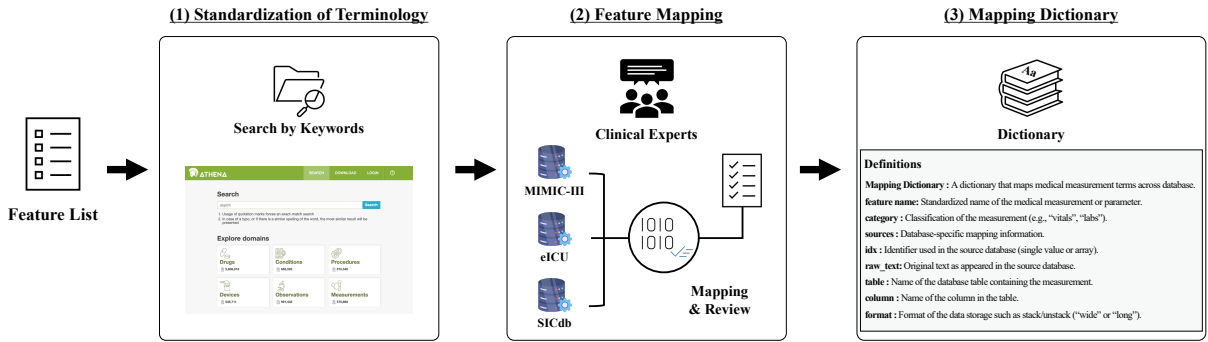


Figure A.3: Illustration of the feature mapping procedure.

cause a single feature can be represented by multiple codes or names, the total number of distinct codes corresponding to these 56 features is 115 in MIMIC-III, 52 in eICU, and 132 in SICdb. These counts exclude cases where a requested feature does not exist in a given database. As shown in Table A.3, some features are absent in certain databases, resulting in true negatives or false positives during evaluation. For absence-aware valuation, balanced accuracy is defined as $bAcc = (TPR + TNR)/2$. A true negative corresponds to a concept that is absent from the database and is correctly predicted as having no valid mapping.

For mapping codes stored as columns, the prediction must include both the table name and column name (e.g., vitalperiodic.temperature, vitalperiodic.systemicsystolic). For codes stored as rows, the prediction must include both the code number and feature name (e.g., (656, Glukose (BGA)), (348, Glukose (ZL))) for MIMIC-III and SICdb. In eICU, where code numbers are not avail-

able, only the feature name is used for code mapping evaluation.

A.4 Curated Database Notes

For both **Cohort and Feature Selection Evaluation** and **Code Mapping Evaluation**, *PreCISE-EMR* includes curated database notes for each EMR database. These notes summarize concise database-level caveats identified by clinical experts, such as excluded tables, time-offset conventions, definition-table choices, and code-identifier conventions. They were created before evaluation-set construction and are not task-specific SQL templates, executable extraction pipelines, or gold mapping dictionaries. All methods, including baselines, receive the same curated notes, database manuals, schema metadata, and sample-value schema information. The curated notes are shown in Fig. A.4.

Table A.3: Feature list used for feature mapping in the framework evaluation set. We explored features using the Observation Source Table in the HiRID dataset (Faltys et al., 2021) defined with standard terminology as a reference, and added features that are commonly used in laboratory tests but not included in HiRID. The newly added features were mapped to standard terminology in Athena OHDSI. Additionally, we limited features to vital signs and laboratory tests, and finally selected features that exist in at least one of the three datasets, resulting in a total of 56 features.

Feature	MIMIC-III	eICU	SICdb
Core body temperature	✓	✓	✓
Heart rate	✓	✓	✓
Invasive diastolic arterial pressure	✓	✓	✓
Invasive mean arterial pressure	✓	✓	✓
Invasive systolic arterial pressure	✓	✓	✓
Non-invasive diastolic arterial pressure	✓	✓	✓
Non-invasive mean arterial pressure	✓	✓	✓
Non-invasive systolic arterial pressure	✓	✓	✓
Respiratory rate	✓	✓	✓
Alanine aminotransferase [Enzymatic activity/volume] in Serum or Plasma	✓	✓	✓
Albumin [Mass/volume] in Serum or Plasma	✓	✓	✓
Alkaline phosphatase [Enzymatic activity/volume] in Blood	✓	✓	✓
aPTT in Blood by Coagulation assay	✗	✓	✓
Aspartate aminotransferase [Enzymatic activity/volume] in Serum or Plasma	✓	✓	✓
Band form neutrophils/100 leukocytes in Blood	✓	✗	✓
Base excess in Arterial blood by calculation	✓	✓	✓
Bicarbonate [Moles/volume] in Arterial blood	✓	✓	✓
Bilirubin.direct [Mass/volume] in Serum or Plasma	✓	✓	✓
Bilirubin.total [Moles/volume] in Serum or Plasma	✓	✓	✓
C reactive protein [Mass/volume] in Serum or Plasma	✓	✓	✓
Calcium [Moles/volume] in Blood	✓	✓	✓
Calcium.ionized [Moles/volume] in Blood	✓	✓	✓
Carbon dioxide [Partial pressure] in Arterial blood	✓	✓	✓
Chloride [Moles/volume] in Blood	✓	✓	✓
Cholesterol [Mass/volume] in Serum or Plasma	✓	✓	✓
Creatine kinase [Mass/volume] in Blood	✓	✓	✓
Creatine kinase.MB [Mass/volume] in Blood	✓	✓	✗
Creatine kinase.MB [Mass/volume] in Serum or Plasma	✗	✗	✓
Creatinine [Moles/volume] in Blood	✓	✓	✓
Fibrinogen [Mass/volume] in Platelet poor plasma by Coagulation assay	✓	✓	✓
Glucose [Moles/volume] in Serum or Plasma	✓	✓	✓
Hematocrit [Volume Fraction] of Blood	✓	✓	✓
Hemoglobin [Mass/volume] in Arterial blood	✓	✓	✓
INR in Blood by Coagulation assay	✓	✓	✗
Lactate [Mass/volume] in Arterial blood	✓	✓	✓
Leukocytes [# /volume] in Blood	✓	✗	✓
Lymphocytes [# /volume] in Blood	✓	✓	✓
Magnesium [Moles/volume] in Blood	✓	✓	✓
MCH - Mean corpuscular haemoglobin	✓	✓	✓
MCHC [Mass/volume]	✓	✓	✓
MCV [Entitic volume]	✓	✓	✓
Methemoglobin/Hemoglobin.total in Arterial blood	✓	✓	✓
Neutrophils/100 leukocytes in Blood	✓	✓	✓
Oxygen [Partial pressure] in Arterial blood	✓	✗	✓
Oxygen measurement, partial pressure, arterial	✓	✓	✓
Oxygen saturation in Arterial blood	✓	✓	✓
Partial thromboplastin time ratio	✓	✓	✗
pH of Arterial blood	✓	✓	✓
Phosphate [Moles/volume] in Blood	✓	✓	✓
Platelets [# /volume] in Blood	✓	✓	✓
Potassium [Moles/volume] in Blood	✓	✓	✓
Sodium [Moles/volume] in Blood	✓	✓	✓
Troponin I measurement	✓	✓	✓
Troponin T.cardiac [Mass/volume] in Serum or Plasma	✓	✓	✓
Urea [Moles/volume] in Venous blood	✗	✗	✓
Urea nitrogen [Mass/volume] in Serum or Plasma	✓	✓	✗

B Further Analysis of EMR-AGENT

B.1 Reference-Based Residual-Work Proxy

This appendix provides a reference-based proxy for residual correction effort during EMR onboarding. It is not a user study or a measurement of elapsed-time savings; all generated outputs still require expert review.

Residual Correction Proxy for Cohort & Feature Selection (CFS) Onboarding a new EMR requires clinicians to interpret schema structures,

write SQL logic, and verify extracted feature values. We use reference-output discrepancies as a proxy for cases that may require deeper correction beyond routine verification.

Residual human effort in CFS is quantified using strict correctness measures:

- **Cohort Logic Errors:** A cohort clause is marked incorrect if even a single patient differs from the reference output. Any discrepancy indicates a semantic mismatch that requires full clinician re-validation.

(A) MIMIC-III <i>Prior knowledge for cohort</i> 1. Exclude all tables and values related to Apache. Apache scores record the worst patient state within the first 24 hours after ICU admission. 2. Use only values from the chartevents and labevents tables. 3. To get the patient's age as an integer, use icu intime columns and birth year. Do not use current date, but use year and month information to calculate patient age. <i>Prior knowledge for feature</i> 1. Exclude all tables related to Apache. Apache scores record the worst patient state within the first 24 hours after ICU admission. 2. For Lab-Test features, use only D_LABITEMS table as definition table. 3. For Vital-Sign features, use only D_ITEMS table as definition table.	(B) eICU <i>Prior knowledge for cohort</i> 1. Note that all time information is recorded as minute-level offsets relative to the ICU admission time. 2. Thus for hour-level features, you need to prepare the data by converting the minute-level offsets to hour-level offsets. 3. Unitvisitnumber or patienthealthsystemstayid is NOT the patient's lifetime admission count but the number of ICU stays during one hospital admission (uniquepid is for lifetime admission count). 4. The age column is stored as a string (or numeric but needs validation), cast it to text and apply a regular expression check. 5. Apache related tables are all excluded to avoid confusion. Thus do not consider any apache related tables. <i>Prior knowledge for feature</i> 1. Do not use any apache related tables such as apacheapsvar. 2. Do not use nursecharting, respiratorycharting, and physicalexam tables 3. For definition table search, use only "lab" table and do not reference any other tables. 4. Do not use 'labid or patientunitstayid' as 'unique feature identifier' because it is a row-level event-level IDs. 5. For time-series feature schema linking, use only "vitalperiodic" table for vitalsign (invasive) features 6. For time-series feature schema linking, Use only "vitalaperiodic" table for vitalsign (non-invasive) features 7. For time-series feature schema linking, use only "lab" table for laboratory features.
(C) SICdb <i>Prior knowledge for cohort</i> 1. None <i>Prior knowledge for feature</i> 1. For vital-sign features, use only the data_float_h table and d_references as the definition table. 2. Out of many ids, we picked 'dataid' column as the vital sign mapping code. 3. For Lab-Test features, always include laboratory table as definition table. 4. Out of many ids, we picked 'laboratoryid' column as the Lab-Test mapping code.	

Figure A.4: Curated database notes used as a concise note highlighting dataset guideline identified by clinical experts.

- **Feature Logic Errors:** If any extracted value for gender, age, or LOS is incorrect for any patient, the entire feature clause is considered erroneous, as such errors typically reflect incorrect joins, filters, or temporal logic.

These error categories serve as a reference-based proxy for residual correction burden. In practice, clinicians must still inspect every generated SQL clause because reference outputs are unavailable during deployment and successful execution does not guarantee semantic correctness. A discrepancy with the benchmark reference indicates a case in which review would need to proceed beyond routine verification to correction of joins, filters, or temporal logic.

Each database includes 70 CFS tasks, for a total of 210 outputs across MIMIC-III, eICU, and SICdb. Across these tasks, EMR-AGENT generates 155 logic errors, compared with 339–416 errors for the baselines, corresponding to a 54–63% reduction in reference-identified logic-error cases. We interpret this difference as a reduction in the proxy count of cases requiring deeper correction, rather than as a measured reduction in clinician time or labor.

Residual Code-Search Proxy for Code Mapping (CM) Code Mapping requires extensive manual search effort. We distinguish between low-cost (TP/FP) and high-cost (FN/TN/Remaining) clinician operations. High-cost operations correspond to searching thousands of raw EMR codes, which

dominate real-world preprocessing time. Thus, the total number of FN + TN + Remaining concepts serves as a proxy for high-cost manual search burden. “Remaining” denotes unmapped concepts requiring manual search and is therefore counted with FN/TN operations in this proxy metric.

We evaluate CM over 56 clinical concept types. Because one concept can correspond to multiple raw codes, each EMR contains different numbers of raw measurement codes (52 in eICU, 115 in MIMIC-III, and 132 in SICdb), and some concepts are absent in each EMR (maximal TN = 5, 3, and 4, respectively). For robustness, the CM evaluation is repeated three times (three independent runs) using the same set of clinical concepts, yielding a total of 933 concept-level mapping checks across the three EMRs: $3 \times (52 + 115 + 132 + 5 + 3 + 4)$.

EMR-AGENT reduces high-cost operations from 933 to 399, a 57% reduction in the most labor-intensive part of EMR onboarding. Baselines reduce only 5 to 17%. This metric requires no assumptions about annotation time, but reflects the practical asymmetry that FN/TN searches are more labor-intensive than TP/FP verification. To avoid trivial over-predicting, this evaluation metric must be interpreted with predictive accuracy (Table 1), preventing degenerate solutions.

Overall Impact on Human-in-the-Loop Workload Overall, these analyses quantify reference-based residual correction and search cases rather

Table B.1: **Human correction requirements for Cohort & Feature Selection (CFS)**. EMR-AGENT produces substantially fewer erroneous SQL clauses than all baselines.

Method	Cohort Logic Errors		Feature Logic Errors	
	Correct Cohort	Erroneous Cohort	Correct Feature	Erroneous Feature
Ours (CFSA)	157	53	108	102
ICL (PLUQ)	57	153	11	199
ICL (SeqSQL)	4	206	0	210
DinSQL	15	195	3	207
REACT	54	156	27	183

Table B.2: **Human-labor reduction in Code Mapping (CM)**. High-cost operations approximate the manual search burden required for unresolved or missed concepts.

Method	FN	TN	Remaining [†]	High-Cost Ops	Reduction vs. Human
Human (baseline)	0	0	933	933	–
Ours (CMA)	340	26	33	399	57% ↓
ICL (PLUQ)	792	17	82	891	5% ↓
REACT	652	31	87	770	17% ↓

[†]Remaining = 933 – TP – FN – TN.

than actual reductions in clinician workload or elapsed time.

B.2 Observation-SQL Analysis

Figure B.1 compares the number of observation SQL queries and F1 scores under document ablations. The analysis shows that additional database probing can partially compensate for missing documents, but performance still degrades when schema/document grounding is removed.

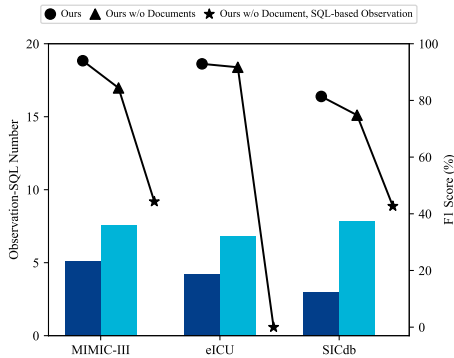


Figure B.1: Comparison of Observation-SQL Number and F1 Score across EMR databases.

B.3 Earlier SICdb-Only Backbone Sensitivity

Table B.3 reports an earlier SICdb-only backbone-sensitivity analysis conducted before the Qwen3.5-27B experiments, comparing Qwen2.5-72B (Yang et al., 2025), Llama-3.1-70B (Grattafiori et al., 2024), Claude-3.5-Haiku

and Claude-3.5-Sonnet (Anthropic, 2024a,b), and Claude-3.7-Sonnet (Anthropic, 2025). These results are included as historical context, showing that older open-weight backbones were substantially weaker before the Qwen3.5-27B transfer experiment.

B.4 Qwen3.5-27B Full Results

Table B.4 provides the complete Qwen3.5-27B results underlying Table 4. It reports secondary metrics and average LLM-call counts for both CFSA and CMA. We omit runtime because it is affected by local serving conditions, batching, and database execution overhead, whereas LLM-call counts more directly reflect the interaction budget of each framework.

B.5 Cost-Performance Comparisons

Unless otherwise specified, all methods are evaluated under the same task-specific maximum LLM-call budget: 30 calls for CFSA and 60 calls for CMA. If a run reaches this budget without producing a valid output, it is terminated and counted as a failure. Reported “Calls” values denote the realized average number of LLM calls before success or termination.

Cost-Performance Analysis for CFS Across Baselines (Table B.5). EMR-AGENT is more cost-efficient than ReAct: although ReAct makes more LLM calls and has longer runtime across

Table B.3: Earlier SICdb-only sensitivity analysis across LLM backbones. Values report mean $\pm$ standard error.

Metric	Qwen2.5-72B	Llama-3.1-70B	Claude-3.5-haiku	Claude-3.7-Sonnet	Claude-3.5-Sonnet
CFSA F1	0.22 $\pm$ 0.05	0.18 $\pm$ 0.04	0.74 $\pm$ 0.05	0.80 $\pm$ 0.05	0.81$\pm$0.04
CFSA Acc.	0.20 $\pm$ 0.04	0.17 $\pm$ 0.04	0.69 $\pm$ 0.04	0.77 $\pm$ 0.03	0.79$\pm$0.04
CMA F1	0.31 $\pm$ 0.01	0.14 $\pm$ 0.02	0.44 $\pm$ 0.00	0.63$\pm$0.02	0.54 $\pm$ 0.03
CMA bAcc.	0.16 $\pm$ 0.01	0.09 $\pm$ 0.01	0.35 $\pm$ 0.00	0.39$\pm$0.01	0.38 $\pm$ 0.02

Table B.4: Full Qwen3.5-27B results for CFSA and CMA. F1, feature accuracy, and balanced accuracy are reported as mean(SE); Calls denotes the average number of LLM calls.

(a) CFSA: ID-F1, feature accuracy, and average LLM calls									
Method	MIMIC-III			eICU			SICdb		
	F1	Acc.	Calls	F1	Acc.	Calls	F1	Acc.	Calls
EMR-AGENT	0.934(.002)	0.962(.001)	8.5	0.890(.004)	0.964(.003)	11.4	0.992(.000)	0.835(.003)	10.2
CHES-Adapted	0.731(.013)	0.708(.012)	4.0	0.891(.008)	0.924(.008)	3.9	0.397(.015)	0.308(.011)	4.0
CHASE-SQL-adapted	0.932(.003)	0.955(.002)	4.0	0.902(.008)	0.927(.008)	4.0	0.429(.016)	0.308(.011)	4.1
EHRAgent-adapted	0.654(.014)	0.652(.014)	4.1	0.357(.015)	0.356(.015)	3.1	0.071(.008)	0.054(.006)	14.0
(b) CMA: F1, balanced accuracy, and average LLM calls									
Method	MIMIC-III			eICU			SICdb		
	F1	bAcc.	Calls	F1	bAcc.	Calls	F1	bAcc.	Calls
EMR-AGENT	0.578(.000)	0.302(.000)	32.4	0.593(.001)	0.324(.001)	6.7	0.603(.000)	0.396(.000)	21.7
CHES-Adapted	0.402(.000)	0.204(.000)	4.0	0.072(.000)	0.055(.000)	4.0	0.538(.000)	0.231(.000)	4.0
CHASE-SQL-adapted	0.257(.000)	0.101(.000)	4.1	0.500(.000)	0.264(.000)	4.1	0.259(.000)	0.104(.000)	4.0
EHRAgent-adapted	0.019(.000)	0.008(.000)	43.8	0.238(.000)	0.129(.000)	28.8	0.000(.000)	0.036(.000)	48.0

all datasets, it consistently achieves substantially lower F1 accuracy. This indicates that EMR-AGENT’s gains are not due to calling the model more frequently. Compared with lightweight baselines such as PLUQ, SeqSQL, and DinSQL, EMR-AGENT uses a moderate number of additional calls but yields large improvements in accuracy. These added calls correspond to *SQL-Observation* and *Error Feedback* steps. Table 2 in the main paper shows that removing these steps causes severe accuracy degradation, confirming that the extra calls are necessary for resolving schema ambiguity, not inefficient recursion.

Cost-Performance Analysis for CM Across Baselines (Table B.6). Across all datasets, EMR-AGENT achieves substantially higher F1 performance compared to PLUQ and ReAct. While PLUQ has the lowest runtime due to being a single-shot method, its accuracy remains near-zero across datasets. ReAct performs multiple iterative steps but still underperforms both in accuracy compared to EMR-AGENT. Overall, EMR-AGENT is not uniformly cheaper than stronger Text-to-SQL agents. Under Qwen3.5-27B, CHES and CHASE-SQL use approximately four CFSA calls on MIMIC-III and eICU, compared with 8.5 and 11.4 calls for EMR-AGENT, while achieving

comparable CFSA performance on these reference databases. We therefore interpret the results as an accuracy-interaction trade-off: the additional structured interactions are most beneficial for CMA and SICdb CFSA rather than uniformly reducing inference cost.

B.6 General Coding-Agent Baseline

To test whether a general-purpose coding agent can perform the same tasks without EMR-AGENT’s task-specific orchestration, we additionally evaluate Aider v0.82.3 using the same Qwen3.5-27B backbone, schema metadata, database documents, curated notes, read-only SQL access, tasks, and scoring protocol. Aider runs its native loop without an LLM-call cap under a 600-second timeout per task. Each task uses a single target file (solve.py) with `-yes-always`, `-no-auto-commits`, `-map-tokens 0`, and `-auto-test`; the evaluation harness is `aider_baseline.py`.

As shown in Table 10, EMR-AGENT outperforms Aider on both CFSA and CMA across all three databases. Aider produced no usable output on 14% of CFSA tasks and 17% of CMA tasks, with observed failures including unrecovered SQL execution errors and code selection without systematic enumeration of database-local candidate

Table B.5: **Cohort and Feature Selection (CFS)**: LLM call count, runtime (second), and F1 score across MIMIC-III, eICU, and SICdb.

Model	MIMIC-III			eICU			SICdb		
	Calls	Runtime	F1	Calls	Runtime	F1	Calls	Runtime	F1
Ours (CFSA)	9.98	115.28	0.94	10.31	124.84	0.929	7.02	69.80	0.814
ICL (PLUQ)	1	15.55	0.749	1	16.88	0.132	1	10.72	0.407
ICL (SeqSQL)	2.85	30.41	0.040	2.85	27.58	0.000	2.85	20.28	0.040
DinSQL	4	42.72	0.726	4	38.42	0.000	4	38.30	0.071
ReAct	11.17	182.14	0.308	15	266.54	0.524	10.46	72.86	0.503

Table B.6: **Code Mapping (CM)**: LLM call count, runtime (second), and F1 score across MIMIC-III, eICU, and SICdb.

Model	MIMIC-III			eICU			SICdb		
	Calls	Runtime	F1	Calls	Runtime	F1	Calls	Runtime	F1
Ours (CMA)	17.48	72.97	0.516	16.26	53.02	0.648	18.46	73.11	0.536
ICL (PLUQ)	1	12.92	0.022	1	9.88	0.125	1	8.33	0.119
ReAct	9.17	21.80	0.214	4.47	64.82	0.067	9.17	16.84	0.218

Table 10: Same-backbone comparison with Aider using Qwen3.5-27B. Values report mean $\pm$ SE F1.

Task	Method	MIMIC-III	eICU	SICdb
CFSA	Aider	0.719 $\pm$.006	0.691 $\pm$.018	0.938 $\pm$.008
	EMR-AGENT	0.934$\pm$.002	0.890$\pm$.004	0.992$\pm$.000
CMA	Aider	0.390 $\pm$.022	0.161 $\pm$.035	0.551 $\pm$.004
	EMR-AGENT	0.578$\pm$.000	0.593$\pm$.001	0.603$\pm$.000

Table 11: Additional ablations under Qwen3.5-27B. Values report mean $\pm$ SE F1.

Task	Setting	MIMIC-III	eICU	SICdb
CFSA	Full	0.934 $\pm$.002	0.890 $\pm$.004	0.992 $\pm$.000
CFSA	w/o Curated Notes	0.910 $\pm$.005	0.907 $\pm$.013	0.999 $\pm$.001
CMA	Full	0.578 $\pm$.000	0.593 $\pm$.001	0.603 $\pm$.000
CMA	w/o Curated Notes	0.577 $\pm$.000	0.588 $\pm$.001	0.493 $\pm$.000
CFSA	Full	0.934 $\pm$.002	0.890 $\pm$.004	0.992 $\pm$.000
CFSA	w/o DB Interaction	0.914 $\pm$.039	0.760 $\pm$.021	0.812 $\pm$.014

rows.

B.7 Curated-Notes and DB-Interaction Ablations

We further isolate two sources of performance under Qwen3.5-27B. The curated-notes ablation removes only the curated database notes while retaining official documents, schema metadata, and live SQL access. The DB-Interaction ablation removes both SQL-based Observation and Error Feedback from CFSA.

Removing curated notes has little effect on MIMIC-III and eICU, although SICdb CMA decreases from 0.603 to 0.493. Removing DB Interaction causes larger CFSA degradation, particularly on eICU and SICdb.

Table 12: Non-LLM retrieval baselines for CMA. Retrieval methods use oracle mapping cardinality. EMR-AGENT results are shown as Claude-3.5-Sonnet / Qwen3.5-27B.

Database	Lexical	Retrieval+rank	EMR-AGENT
MIMIC-III	0.365	0.333	0.516 / 0.578
eICU	0.623	0.604	0.648 / 0.593
SICdb	0.345	0.414	0.536 / 0.603

B.8 Non-LLM Retrieval Baselines for CMA

We additionally compare CMA with two non-generative baselines: character- n -gram TF-IDF lexical retrieval and retrieval followed by reranking with Qwen3-Embedding-8B (rerank_cma.py). Because reference mappings can be one-to-many, we use a favorable oracle-cardinality setting in which each method returns the top- k candidates, with k equal to the number of reference mappings for that concept.

Retrieval is competitive on eICU, but remains below EMR-AGENT on MIMIC-III and SICdb even under oracle mapping cardinality.

C Case Studies of Agent Reasoning

This appendix provides a concise walkthrough illustrating how EMR-AGENT performs Cohort/Feature Selection (CFSA) and Code Mapping (CMA). Each figure presents the key steps taken by each agent.

C.1 Cohort and Feature Selection Agent (CFSa) Case Study

In Figure C.1, the agent begins by interpreting the user instructions together with the database metadata. It then performs schema linking (mapping the instruction to relevant tables and columns in SICdb) and retrieves the corresponding schema guidelines (e.g., what information is missing to fulfill the user request, the meaning of each column, etc.). In the Figure C.1, the agent identifies key elements such as the **sex column**, **d_reference table**, and **caseid column**.

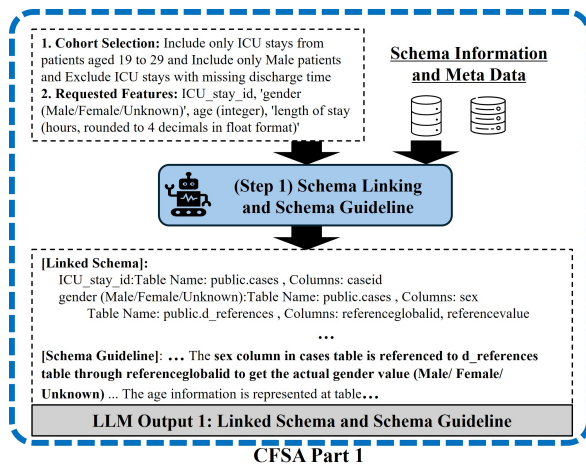


Figure C.1: **Step 1: Schema Linking and Schema Guideline.** The agent identifies tables/columns for ICU_stay_id, gender, age, and length of stay, and constructs an initial schema guideline.

In Figure C.2, after linking the relevant columns, the agent evaluates whether additional information from the EMR is required. Because the gender-code pairing is not yet known, the agent decides that further observation is necessary and issues a SQL-observation query to inspect the actual stored values. Through this inspection, the agent discovers that SICdb encodes gender using numeric reference IDs (735 for male and 736 for female). The agent also figures out that the gender code must be resolved via a join with **d_references**. Recognizing this as essential schema information, the agent updates the schema guideline accordingly.

In Figure C.3, With the mappings and filters resolved, the agent composes the final SQL query satisfying all constraints: age between 19–29, male-only, and non-null LOS, with LOS converted from seconds to hours. If the SQL execution produced a syntax error, it would feedback to SQL-generation step with the error message. If the SQL execu-

tion produced a schema mismatch error (e.g., missing columns, invalid types), it would feedback to Schema Linking and Schema Guideline step with the error message.

C.2 Code Mapping Agent (CMA) Case Study

In Figure C.4, the agent begins by examining **d_labitems** and **labevents**, together with the MIMIC-III metadata, to understand how laboratory measurements are defined and stored in MIMIC-III. This step establishes the structural context necessary to search for the requested feature, *Hemoglobin [Mass/Volume] in Arterial Blood*, before attempting any candidate matching or Feature Locating.

In Figure C.5, after the initial schema linking, the agent checks whether the target feature, *Hemoglobin [Mass/volume] in Arterial blood*, appears literally in any column names. Because neither the full name nor any similar name matches the target concept, the agent concludes that literal matching is not feasible. Based on this assessment, it proceeds to the Candidate Matching stage to search for the target item.

In Figure C.6, the agent performs a controlled SQL query to retrieve all candidate laboratory items and then applies LLM-based semantic matching over the resulting metadata. Rather than relying on raw lexical similarity, the agent compares item-level attributes such as code number, label, unit type, and category to determine which entries correspond to the intended Hemoglobin measurement. Through this semantic reasoning process, the agent narrows the set of Hemoglobin-related itemids candidates.

C.3 Qualitative Failure Analysis

We further inspect representative baseline failures and failure boundaries of EMR-AGENT under the Qwen3.5-27B setting. These examples illustrate why live database observation and explicit candidate matching are useful, while also showing errors that execution feedback alone cannot detect.

Baseline failure: SICdb value encoding. For the CFSa request to select male ICU stays aged 19–29 while excluding stays with missing discharge time, CHES and CHASE-SQL generated filters using `d_references.referencevalue = 'Male'` without first inspecting the stored values. SICdb stores the corresponding labels in lowercase (male, female; codes 735 and 736), so these queries exe-

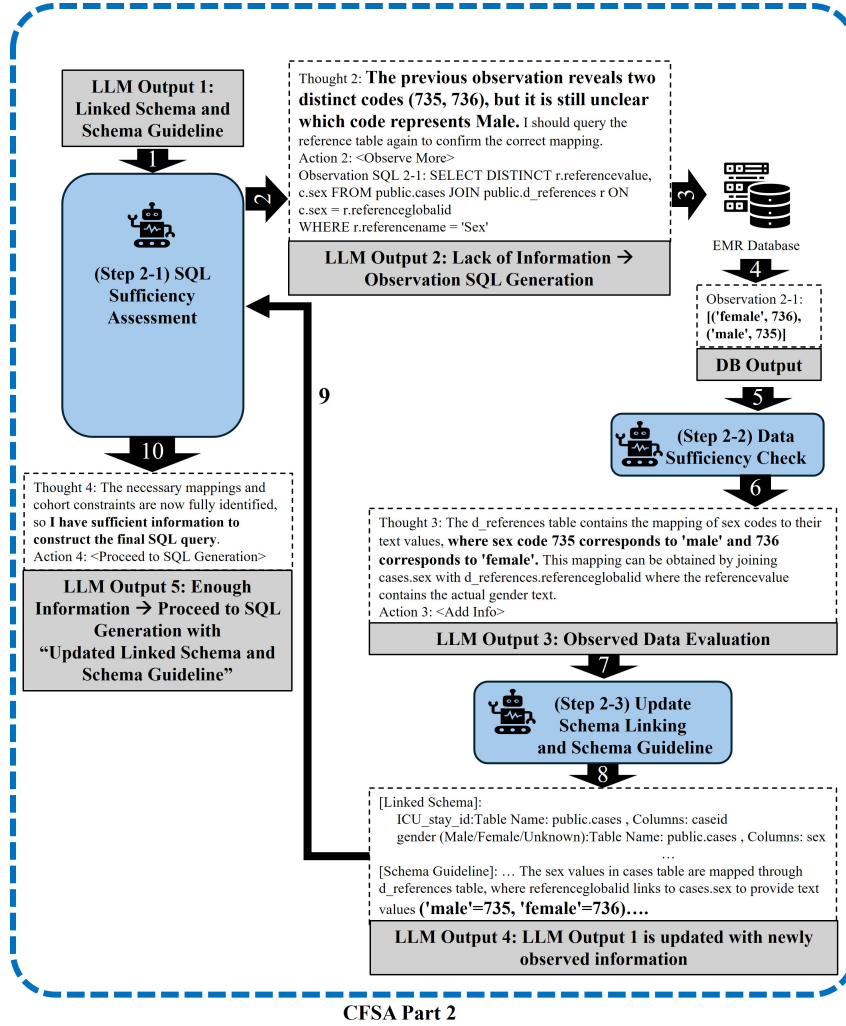


Figure C.2: **Step 2: SQL Observation.** Observation queries reveal the true gender mapping: 735 = Male, 736 = Female. The agent updates its schema guideline and confirms sufficiency for SQL generation.

cuted without error but returned zero rows. In contrast, EMR-AGENT first queried the database and observed `[('female', 736), ('male', 735)]` (as illustrated in Fig. C.2), and subsequently used the observed lowercase value.

Baseline failure: eICU type handling. In eICU, the `patient.age` column is stored as text and contains non-numeric values such as `>89` and empty strings. In an EHRAgent run, directly applying `CAST(age AS INTEGER)` resulted in invalid input syntax for type integer, while applying `ROUND` directly to a double-precision expression also caused a PostgreSQL type error. EMR-AGENT instead inspected the distinct stored age values and validated numeric strings, e.g., `age ~ '^[0-9]+$'`, before casting.

Baseline failure: MIMIC-III code mapping. For several MIMIC-III concepts, CHASE-SQL re-

Table 13: Representative MIMIC-III CMA predictions under Qwen3.5-72B.

Requested concept	CHASE-SQL	EMR-AGENT
Troponin T	51222 (Hemoglobin)	51003 (Troponin T)
C-reactive protein	51222 (Hemoglobin)	50889 (CRP)
ALT	50862 (Albumin)	50861 (ALT)
Oxygen saturation	50812 (Intubated/status)	50817 (Oxygen saturation)

turned plausible but incorrect local item codes, whereas EMR-AGENT retrieved and matched database-local candidate rows. Representative examples are shown in Table 13.

EMR-AGENT failure: semantically incorrect derivation. EMR-AGENT can generate SQL that executes successfully but follows an incorrect clinical convention. In one CFSA run, the correct cohort was selected, but length of stay was derived using hospital-level rather than ICU-level timestamps. Because the SQL executed successfully,

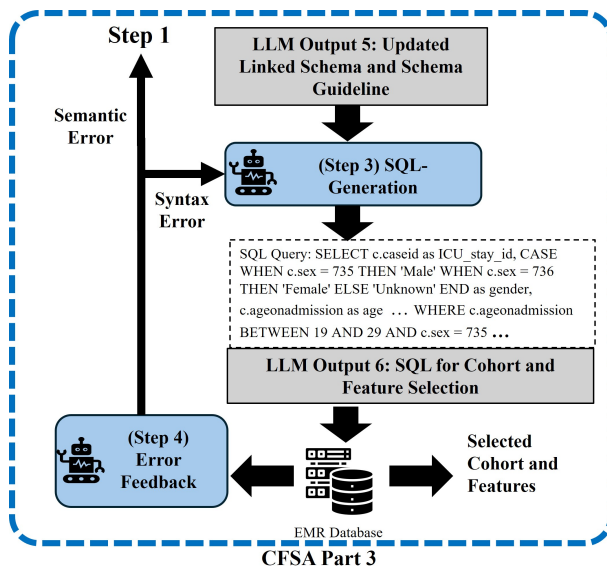


Figure C.3: **Step 3: Final SQL Generation.** The agent produces a correct SQL query with all filters applied and LOS properly converted and rounded.

Error Feedback was not triggered, showing that execution feedback cannot detect all semantic errors.

EMR-AGENT failure: related but non-native identifier. In CMA, EMR-AGENT can identify the clinically relevant concept while returning a standard identifier rather than the required database-native code. In one observed failure, CMA returned a related LOINC identifier instead of the database-native item ID. Such cases motivate expert validation of all proposed mappings.

D Extended Limitations and Broader Implications

Limitation EMR-AGENT is not designed to fully replace human expertise in EMR preprocessing. While it significantly automates data extraction, it remains a supportive tool that requires validation by qualified professionals to ensure accuracy. Unlike hard-coded pipelines that are specifically tailored to individual datasets and can achieve near-perfect accuracy, EMR-AGENT may not consistently reach this level of precision. As a result, data extracted by the agent may require further validation before being used in time-series tabular model training.

Our evaluation focuses on cohort selection scenarios designed to ensure clear, reproducible ground truth across the three ICU datasets. More complex clinical conditions (e.g., sepsis, ventilation) were not included because their definitions

vary across institutions and lack dataset-wide consensus standards. We plan to extend the benchmark to compound and temporal cohort definitions in future work.

A similar limitation applies to our code mapping evaluation. Our current setup focuses on common laboratory tests, which represent one of the few clinical domains where clear and reproducible reference standards exist. More complex concepts such as medications and ventilation parameters would require integration of broader reference terminologies (e.g., RxNorm, SNOMED-CT), dose/route normalization, handling of time-varying concepts, and development of appropriate evaluation frameworks. While EMR-AGENT’s modular architecture is designed to accommodate such extensions, establishing reliable reference standards for these domains remains an open challenge, and we plan to address these limitations in future work.

Furthermore, it is important to acknowledge that EMR-AGENT may carry forward existing biases present in the raw EMR data. These biases, which often stem from historical healthcare inequities and varying data collection practices across different demographic groups, may appear in various forms, such as disparities in demographics, diagnoses, or treatments embedded within the EMR databases. The automated extraction process, while efficient, does not inherently address or mitigate these systematic biases, which can subsequently influence downstream machine learning models. Researchers utilizing EMR-AGENT should be aware of these limitations and implement appropriate strategies for bias detection and fairness assessment in their analyses.

Nevertheless, this agent-based approach introduces a scalable and adaptive paradigm with promising potential for future improvements.

Broader Impacts EMR-AGENT has the potential to reduce the manual workload for clinical experts in managing complex EMR data. However, given the sensitive nature of healthcare information, its deployment must be accompanied by rigorous validation and ongoing oversight to ensure safety, accuracy, and ethical compliance.

Although EMR-AGENT utilizes large language models for database interactions, its environmental impact remains relatively modest as it operates solely during inference, without the need for training or fine-tuning. Moreover, from the perspective of a broader research community, the frame-

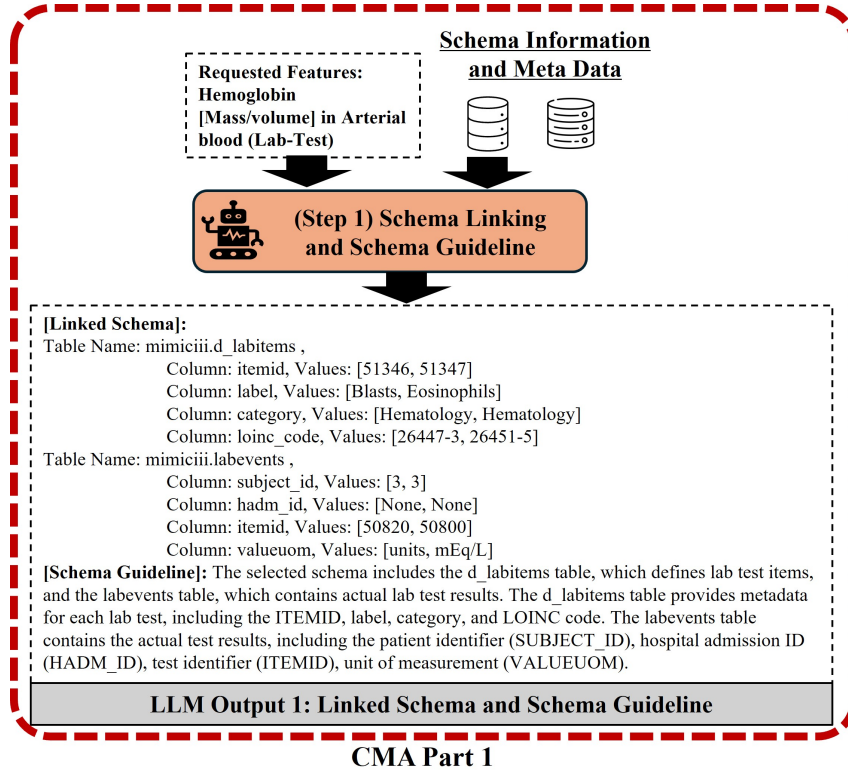


Figure C.4: **Step 1: Schema Linking for Laboratory Measurements.** The agent selects and inspects **d_labitems** and **labevents**, including item labels, units, fluid types, categories, and LOINC codes, to understand how lab tests are represented in MIMIC-III. This schema-level analysis provides the foundation for locating the user-requested feature *Hemoglobin [Mass/Volume] in Arterial Blood*.

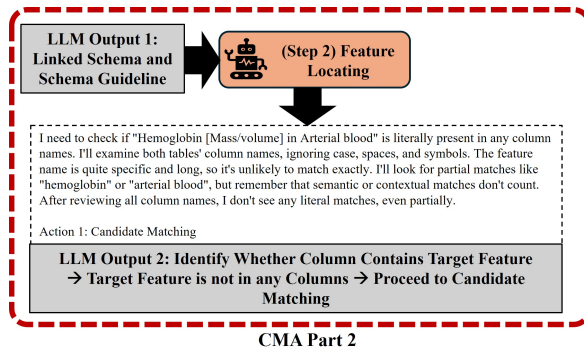


Figure C.5: **Step 2: Feature Locating.** The agent verifies that the target concept does not appear as a literal column name in the schema. After confirming the absence of any direct lexical match, the agent transitions to semantic candidate listing, where item-level metadata (e.g., labels, fluids, categories) is examined to identify potential Hemoglobin-related codes.

work offers significant efficiency gains. By providing a standardized and automated solution, EMR-AGENT reduces the need for multiple research teams to develop similar preprocessing pipelines independently, leading to more resource utilization across the community. The framework's reusabil-

ity and scalability further distribute this computational cost across multiple studies and datasets, thereby promoting more standardized and reproducible EMR research practices.

This work is expected to inspire further research in the field and contribute to its advancement, while maintaining a balance between computational efficiency and environmental responsibility.

E Adapted Qwen3.5-27B Baselines

We adapt CHESS (Talaie et al., 2024), CHASE-SQL (Pourreza et al., 2025), and EHRAgent (Shi et al., 2024) to CFSA and CMA because the original systems target Text-to-SQL or EHR question answering rather than joint EMR preprocessing. This appendix specifies the shared setup, adaptation details, and common hyperparameters used for the Qwen3.5-27B comparison in Table 4.

E.1 Shared Setup

All adapted baselines and EMR-AGENT use the same locally deployed Qwen3.5-27B backbone, benchmark inputs, schema metadata with $N=10$ sample values per column, database documents,

Table E.1: Adapted-baseline pipeline summary under Qwen3.5-27B. All methods share the same backbone, benchmark inputs, schema/document context, SQL interface, and task-specific LLM-call budget.

Method	CFSA pipeline	CMA pipeline
CHESS-adapted	Retrieval → schema selection → candidate SQL → execute/select	Retrieval → schema selection → listing SQL → match
CHASE-SQL-adapted	Multi-path SQL generation → execute/select	Multi-path listing/mapping generation → merge
EHRAgent-adapted	Tool loop → final SQL	Tool loop → final mappings
EMR-AGENT	Schema Guideline → SQL Observation → SQL Generation → Error Feedback	Schema Guideline → Feature Locating → Candidate Matching

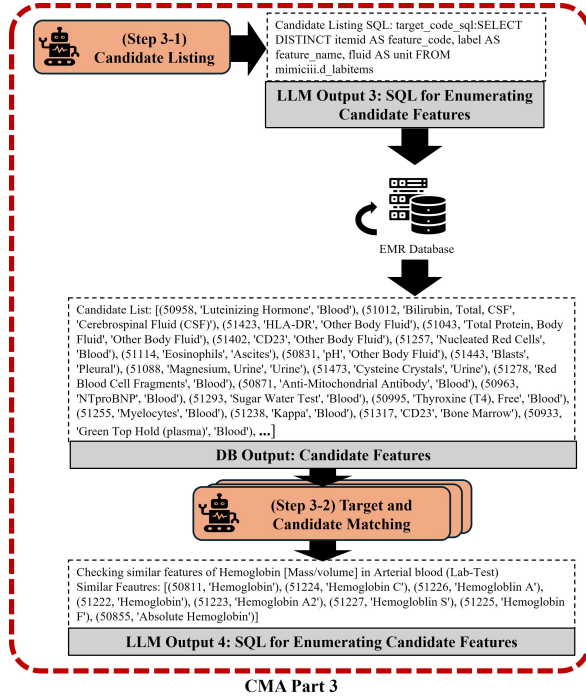


Figure C.6: **Step 3: Candidate Matching.** From retrieved candidates, the agent selects multiple Hemoglobin-related itemids through semantic reasoning over item labels and metadata.

curated notes, foreign-key information, read-only PostgreSQL interface, and task-specific LLM-call budget. The adapted baselines reuse only shared task loading, schema/document loading, SQL execution, evaluation, and logging utilities. They do not use EMR-AGENT’s Schema Guideline, SQL-based Observation, Error Feedback, or CMA Candidate Matching modules. CFSA outputs are parsed as final SQL queries, and CMA outputs are parsed as final item-code or direct-column mappings; malformed outputs receive at most one repair prompt before being counted as failures.

E.2 CHESS-adapted

CFSA. Following CHESS’s retrieval, schema-selection, candidate-generation, and validation

structure, the CFSA adaptation retrieves task-relevant schema/document context, selects a compact sub-schema, generates multiple SQL candidates, executes them through the shared SQL interface, and selects one final SQL query based on execution feedback.

CMA. For CMA, we adapt the same stages to retrieve candidate definition tables, generate candidate-listing SQL, execute the listings, and select item-code or direct-column mappings from the retrieved metadata.

E.3 CHASE-SQL-adapted

CFSA. Following CHASE-SQL’s multi-path candidate-generation and selection design, the CFSA adaptation generates SQL candidates through multiple reasoning paths, including decomposition-style reasoning, execution-plan reasoning, and instance-aware example generation. Valid candidates are executed, and a selector prompt returns the final SQL query.

CMA. For CMA, the same multi-path structure proposes listing SQL queries and mapping candidates. Executed listings are deduplicated, and a selector prompt merges the proposals into one final mapping list.

E.4 EHRAgent-adapted

EHRAgent-adapted is implemented as a tool-using agent with schema inspection and read-only SQL execution tools. At each step, the agent either calls a tool or emits a final answer. For CFSA, the final answer is a SQL query; for CMA, it is an item-code or direct-column mapping list.

E.5 Common Hyperparameters

- Backbone: locally deployed Qwen3.5-27B with temperature=0, and top_p=0.9.
- Maximum LLM-call budget: 30 calls for CFSA and 60 calls for CMA. Runs that reach

the budget without a valid final output are terminated and counted as failures.

- CHESS-adapted candidate pool: $N_c=3$ SQL candidates for CFSA.
- Candidate-listing row cap: 300 rows per listing SQL after SELECT DISTINCT, deduplicated before matching.

E.6 Reproducibility Artifacts

For adapted-baseline runs, we release the prompt templates, parsers, model identifier and decoding settings, per-task LLM-call counts, runtime, candidate-listing SQL strings, and final SQL or mapping outputs. Patient-level rows returned during SQL execution are used only for evaluation and are not serialized into released logs.

F Baselines Prompts

Since the controlled-backbone baselines are not designed for our task, we adapt their prompts from the original methods while preserving each model’s structural format. In this section, we present the prompt settings for PLUQ, SeqSQL, DinSQL, and ReAct. The adapted Qwen3.5-27B baselines CHESS, CHASE-SQL, and EHRAgent are described separately in Appendix E because they are implemented as multi-step agent workflows rather than single prompt templates.

F.1 ICL in PLUQ

Here, the schema information format and prompt style are adopted from PLUQ (Jo et al., 2024). This baseline utilizes the LLM in a single-turn setting only.

Cohort and Feature Selection prompt

You are given Database information and the Question. Generate the PostgreSQL query for the following question. Note that you should generate 'null' if the question cannot be converted to SQL query given information. Get only one SQL query as plain text. Do not include code delimiters (e.g., ““sql), comments, or any additional text.

[Schema Information]: {Schema_information}
[Curated database notes]: {Evaluation_Memo}
[Database Manual]: {Database_Manual}

Q: List all {Feature_Selection} information that satisfy following [Cohort Selection].

[Cohort Selection]: {Cohort_Selection}
Ensure the output in PostgreSQL strictly follows the order and format specified in each () of {Feature_Selection}.
SQL Query:

Code mapping prompt

You are given a [Database schema] and a [Feature].
Task: Analyze the information provided below and classify the feature into one of the following categories:
<get schema>: Select this if you can find a column whose name literally matches any part of the given [Feature].
<get definition SQL>: Select this if no such column exists, but you can retrieve the corresponding feature information using an SQL query. The query should return the unique feature identifier, feature name, and unit from the definition table related to the [Feature].
<null>: Select this if neither a matching schema nor an SQL definition can be found. Instructions:

- If you choose <get schema>, provide the matching table and column in the format: Table_Name.Column_Name
- If you choose <get definition SQL>, provide an SQL query in the format: SELECT unique_feature_identifier, feature_Name, unit FROM dbname.Table_A WHERE ...

[Schema Information]: {Schema_information}
[Curated database notes]: {Evaluation_Memo}
[Database Manual]: {Database_Manual}
[Feature] : {Target_Feature}

Output Format:

<classification>
<get schema>, <get mapping SQL>, or <null>
</classification>
<answer>
[get answer for selected classification formation]
</answer>

F.2 ICL in SeqSQL

SeqSQL (Ryu et al., 2024) is a sequential generation approach for complex SQL queries by decomposing cohort selection into individual conditions. Each decomposed condition’s SQL is generated step-by-step, leveraging the outputs of previous steps to structurally compose the final SQL query. For the Cohort and Feature Selection task, we generate SQL queries corresponding to various conditions and implemented the baseline by combining these conditions using logical conjunctions ("and"). And we utilize all prompt structure from (Ryu et al., 2024) except for 20-shot Examples; Post-processing Detail, SQL-like Rep.Description, Test Question. However, for the Code Mapping task, where the core idea is database search based on a single condition, the use of SeqSQL was unsuitable due to the mismatch in task characteristics, and thus it was not implemented for comparison.

Listing 1: Logic of splitting cohort selection into simple condition in Python

```
question_all = []

question_information = f"List all {requested_features.strip()} information. Ensure the output in PostgreSQL strictly follows the order and format specified in each () of {requested_features.strip()}."
question_all.append(question_information)

cohort_selection = cohort_selection.split("and")
```

```

if isinstance(cohort_selection, list):
    for condition in cohort_selection:
        question_information = f"Retrieve only the cases \"{
            condition.strip()}\\""
        question_all.append(question_information)
else:
    question_information = f"Retrieve only the cases \"{
        cohort_selection}\\""
    question_all.append(question_information)

```

Cohort and Feature Selection prompt

Get only one PostgreSQL query as plain text. Do not include code delimiters (e.g., ““sql), comments, or any additional text.

[Schema Information]: {Schema_information}
 [Curated database notes]: {Evaluation_Memo}
 [Database Manual]: {Database_Manual}

– Post-processing Detail

Please note that:

1. Questions asking whether a specific number falls within a normal range can be formulated as follows and will be changed through the post_processing process.

NLQ: Had the value of result measured during result been normal?

SQL: SELECT COUNT(*)>0 FROM chartevents WHERE chartevents.icustay_id IN (...) AND chartevents.valuenum BETWEEN sao2_lower AND sao2_upper

2. Similarly, for questions that require the current time, we will use ‘current_time’ as a placeholder and adjust it as necessary. For reference, the current time is assumed to be “2105-12-31 23:59:00”. Therefore, if there is the expression “this month” means 2105-12.

– SQL-like Rep. Description

PREV_QUERY and PREV_RESULT tokens allow for referencing and reusing the SQL code and results of previous queries in subsequent ones.

The PREV_QUERY token is used to represent the SQL code of the previous query, essentially allowing the new query to build upon it or modify it. SQL queries can also start with the PREV_QUERY token, which enables the duplication and utilization of the previous query in the new one.

The PREV_RESULT token, on the other hand, is used to represent the example of result set from a previous query, rather than the query itself. This is useful when we want to use the results of a previous query directly within a new query.

– TEST_QUESTION

NLQ1:{question_all[0]}

SQL1:

F.3 DinSQL

DinSQL (Pourreza and Rafiei, 2023) generates SQL queries by selecting the most appropriate schema based on both the database information and the given cohort selection condition. Then it classifies the complexity of the condition and generates SQL by its complex state followed with self-correction mechanism. DinSQL is comparable to our method in its ability to handle complex condition-based SQL generation, making it suitable for comparison in the Cohort and Feature Se-

lection task, it is not appropriate for Code Mapping, which is about database searching for simple, single-condition. Thus DinSQL has been used in Cohort and Feature Selection task.

F.3.1 Cohort and Feature Selection

Schema linking prompt

Find the Schema_links for generating SQL queries for each question based on the [Schema Information], [Curated database notes], and [Database Manual].

[Schema Information]: {Schema_information}
 [Curated database notes]: {Evaluation_Memo}
 [Database Manual]: {Database_Manual}

Q: List all {Target_Features} information that satisfy following [Cohort Selection].

[Cohort Selection]: {Cohort_Selection}

Make PostgreSQL follow the order of the provided information, categories, type.

A: Let’s think step by step.

Classification prompt

For the given question, classify it as EASY, NON-NESTED, or NESTED based on nested queries and JOIN.

if need nested queries: predict NESTED

elif need JOIN and don’t need nested queries: predict NON-NESTED

elif don’t need JOIN and don’t need nested queries: predict EASY

Q: List all {Target_Features} information that satisfy following [Cohort Selection].

[Cohort Selection]: {Cohort_Selection}

Make PostgreSQL follow the order of the provided information, categories, type.

Schema_links: {Schema_links}

A: Let’s think step by step.

SQL generation prompt

easy_prompt

Use the schema links to generate the SQL queries for each of the questions.

Q: "Find the buildings which have rooms with capacity more than 50."

Schema_links: [classroom.building,classroom.capacity,50]

SQL: SELECT DISTINCT building FROM DB_Name.classroom WHERE capacity > 50

...

Q: List all {Target_Features} information that satisfy following [Cohort Selection].

[Cohort Selection]: {Cohort_Selection}

Make PostgreSQL follow the order of the provided information, categories, type.

Schema_links: {Schema_links}

SQL:

medium_prompt

Use the the schema links and Intermediate_representation to generate the SQL queries for each of the questions.

Q: "Find the total budgets of the Marketing or Finance department."
Schema_links: [department.budget, department.dept_Name, Marketing, Finance]
A: Let's think step by step. For creating the SQL for the given question, we need to join these tables = []. First, create an intermediate representation, then use it to construct the SQL query.
Intermediate_representation: select sum(department.budget) from department where department.dept_Name = "Marketing" or department.dept_Name = "Finance"
SQL: SELECT sum(budget) FROM DB_Name.department WHERE dept_Name = 'Marketing' OR dept_Name = 'Finance'
...

Q: List all {Target_Features} information that satisfy following [Cohort Selection].
[Cohort Selection]: {Cohort_Selection}
Make PostgreSQL follow the order of the provided information, categories, type.
Selected_Schema: {Selected_Schema}
A: Let's think step by step.

hard_prompt

Use the intermediate representation and the schema links to generate the SQL queries for each of the questions.

Q: "Find the title of courses that have two prerequisites?"
Schema_links: [course.title, course.course_id = prereq.course_id]
A: Let's think step by step. "Find the title of courses that have two prerequisites?" can be solved by knowing the answer to the following sub-question "What are the titles for courses with two prerequisites?". The SQL query for the sub-question "What are the titles for courses with two prerequisites?" is SELECT T1.title FROM course AS T1 JOIN prereq AS T2 ON T1.course_id = T2.course_id GROUP BY T2.course_id HAVING count(*) = 2 So, the answer to the question "Find the title of courses that have two prerequisites?" is =
Intermediate_representation: select course.title from course where count (prereq.*) = 2 group by prereq.course_id SQL: SELECT T1.title FROM DB_Name.course AS T1 JOIN DB_Name.prereq AS T2 ON T1.course_id = T2.course_id GROUP BY T2.course_id HAVING count(*) = 2
...

Q: List all {Target_Features} information that satisfy following [Cohort Selection].
[Cohort Selection]: {Cohort_Selection}
Make PostgreSQL follow the order of the provided information, categories, type.
Schema_links: {Schema_links}
A: Let's think step by step.

Self-correction prompt

For the given question, use the provided tables, columns, foreign keys to fix the SQL. If correct, return as is.

Question: List all {Target_Features} information that satisfy following [Cohort Selection].
[Cohort Selection]: {Cohort_Selection}
Make PostgreSQL follow the order of the provided information, categories, type.
Schema_links: {Schema_links}
SQL Query: {sql_query}

Fixed SQL Query:
SELECT

F.4 REACT

REACT (Yao et al., 2023) proposes a structured reasoning framework in which an agent takes appropriate actions based on observations from given environment to solve tasks. In our setting, the task involves generating proper SQL to get the user-requested dataset from a fixed database. We extend this structure to both the Cohort and Feature Selection and Code Mapping tasks by formulating SQL generation as a sequence of reasoning steps. At each step, the model performs an action, and observes results from the database by predefined tool, enabling it to iteratively refine its reasoning toward solving the task. We use same prompt in Section F.1.

Listing 2: REACT interacting with database in Python

```
from langchain_core.tools import tool
from langgraph.prebuilt import create_react_agent

def execute_query(query: str):
    """Use this to execute a query against the database."""
    try:
        db_observation = db_connector.connect(query)
    except Exception as e:
        return f"Error executing query: {str(e)}"

    if len(db_observation) > args.max_obsoutput_len:
        db_observation = db_observation[:args.max_obsoutput_len]

    return f"SQL Successfully executed. The example of {args.max_obsoutput_len} rows are as follows:\n{db_observation}"

def react_generation(prompt, llm_model):
    tools = [execute_query]
    react_agent = create_react_agent(model=llm_model, tools=tools)

    agent_inputs = {"messages": [{"user", prompt}]}
    # print(agent_inputs)

    stream = react_agent.stream(agent_inputs, stream_mode="values", config={"recursion_limit": 20})

    response_list = print_stream(stream)
    api_run_count = str(response_list).count('AIMessage')
    observation_count = str(response_list).count('ToolMessage')
    final_result = response_list[-1]["messages"][-1].text()

    return final_result, api_run_count, observation_count
```

G Appendix G. EMR-AGENT PROMPTS

G.1 Prompts of CFSA (Cohort and Feature Selection Agent)

The following provides the detailed prompts used for CFSA, as described in Section 3.2.

G.1.1 Schema Linking and Guideline Generation (Mapping Schema)

Schema Linking and Guideline Generation (for Mapping Schema)

Using [Database schema information], select all schema that are necessary to extract [Features].

- Please select exact table name(s) and column name(s) in [Database schema information].
- Follow the exact "Format" under [Notes] without any extra symbols, code delimiters.

[Notes]:

- Identify only definition schema with mapping information that can be used to extract patients of [Cohort Selection] with [Features].
- Exclude all tables that have actual numeric measurement (vital sign or lab test) columns.
- If identified tables have measurement unit information (not results), get all columns without result information.
- After listing the schema for each feature, provide a [Schema Guideline] in a paragraph of no more than 10 sentences, explaining the details of the columns (such as type or how to interpret the values).
- Output Format:

```
Mapping Table: dbname.Table_A , Columns: Column_a,
Column_b
Mapping Table: dbname.Table_B , Columns: Column_1,
Column_2
[Schema Guideline]: (a paragraph of no more than 10
sentences)
```

[Schema Information]: {Schema_information}
[Curated database notes]: {Evaluation_Memo}
[Database Manual]: {Database_Manual}
[Cohort Selection]: {Cohort_Selection}
[Features]: {Feature_Selection}

G.1.2 Schema Linking and Guideline Generation (Feature Schema)

Schema Linking and Guideline Generation (for Feature Schema)

Using [Database schema information], select all schema that are necessary to extract [requested feature].

- Please select exact table name(s) and column name(s) in [Database schema information].
- Follow the exact "Format" under [Notes] without any extra symbols, code delimiters.

[Notes]:

- Get all schema (tables, columns) related to each element in [Features] and [Cohort Selection].
- After listing the schema for each element in [Features] and [Cohort Selection], provide a [Schema Guideline] in a paragraph of no more than 15 sentences, explaining the details of the selected schema's columns (such as type or example values) and how to generate

SQL to obtain patients from [Cohort Selection] with each [Features] and what it is missing to get the correct result.

- If necessary, utilize [Foreign Key] and [Mapping Table] from [Database schema information] when generating [Schema Guideline] for [Cohort Selection].
- Get patient's related year, date, time information such as admission date, birth date, etc.
- [Feature name] must be exactly same with [Feature].
- Output Format:

```
[Feature name]
Table Name: dbname.Table_A , Columns: Column_a,
Column_b
Table Name: dbname.Table_B , Columns: Column_1,
Column_2
[Feature name]
Table Name: dbname.Table_A , Columns: Column_a,
Column_b
Table Name: dbname.Table_B , Columns: Column_1,
Column_2
...
[Schema Guideline]: (a paragraph of no more than 15
sentences)
```

[Schema Information]: {Schema_information}
[Curated database notes]: {Evaluation_Memo}
[Database Manual]: {Database_Manual}
[Cohort Selection]: {Cohort_Selection}
[Features]: {Feature_Selection}

G.1.3 SQL Sufficiency Assessment

SQL Sufficiency Assessment

You are an assistant tasked with evaluating the provided schema and guideline to determine if they are sufficient to support data extraction requirements.

Carefully review the following components:

- Original Schema: The schema before Schema Linking.
- Selected Schema: The schema and its guideline to assist to extract patients according to [Cohort Selection] with specified features [Target Features].
- Target Features: Specific features required for each patient. Note that names in [Target Features] are not always same in [Schema], do not assume value in schema.
- Cohort Selection: specifications for the configuration of patients to extract.
- Mapping Table: A table(s) and column(s) that contain mapping information of certain features, indicating detail/s/definitions of certain features.
- Foreign Key: A foreign keys of the original schema.
- Error Feedback: If available, feedback from previously generated SQL queries indicating errors.
- Previous Observation (if provided): Previously observed information through SQL queries. Do not generate any SQL query that is already in [Previous Observation].

Task:

Assess whether the current [Selected Schema] and associated [Schema Guideline] are ENOUGH to extract the [Patients] according to [Cohort Selection] with [Target Features]. Classify your evaluation clearly into one of the following:

- <need more information>: The [Selected Schema] and [Schema Guideline] are insufficient or require clarification.
- Do not simply assume the names in [Target Features] and [Cohort Selection] are in [Selected Schema] and [Schema Guideline]. If you are not sure about the values, you need to first check or ask for the actual values that exist in the column (e.g., via 'SELECT DISTINCT column FROM table') before using them.

- Only use a specific value in WHERE clauses if it is explicitly observed in the schema or query result, otherwise keep you position as <need more information>.

- If [Error Feedback] exists and indicates issues, generate additional SQL queries to retrieve missing details. <correct>: The provided [Selected Schema] and [Schema Guideline] are sufficient.

If you classified the schema as <need more information>, - Generate SQL queries to retrieve the necessary additional details.

- If multiple queries are needed, separate each with ||.

- Do not generate SQL queries that retrieve entire tables — focus only on concise, targeted retrievals.

- If you need to use [Mapping Table] and [Foreign Key], please use them in the SQL queries.

Output Format: Provide your response exactly as below, without additional commentary or text:

<think>

[Clearly and concisely explain your reasoning behind the classifications based on the given information.]

</think>

<output>

<need more information> or <correct>

</output>

<SQL queries>

[If you classified the schema as <need more information>, based on you think process, [Schema Guideline] and [Additional Information], provide SQL queries to retrieve additional details from the schema using [Original Schema], [Mapping Table] and [Foreign Key]. If multiple queries are needed, separate each with ||. Note that the number of SQL queries should not exceed [Max SQL Search At Once]. Do not include any SQL query that is already in [Previous Observation].]

</SQL queries>

[Original Schema]:{Original_Schema}

[Selected Schema]:{Selected_Schema}

[Target Features]:{Target_Features}

[Cohort Selection]:{Cohort_Selection}

[Mapping Table]:{Mapping_Table}

[Foreign Key]:{Foreign_Key}

[Previous Observation]:{Previous_Observation}

[Error Feedback]:{Error_Feedback}

- SQL Observation: Results from executed SQL queries, provided as a dictionary (query-output pairs), offering further insights into [Original Schema] and possibly suggest more information to add to [Selected Schema]. The output could be an error message if the SQL query is failed. Keep in mind that the length of [SQL Observation] is limited to 20.

- Pre-Observation: Previously observed information.

Task:

- Select one of the below two options:

<Add info>: If you found something valuable information from [SQL Observation]

<No info>: If you found nothing valuable information from [SQL Observation]

- If you selected <Add info>, provide the gained information from [SQL Observation] in less than 5 sentences between <Add info> and </Add info>. The gained information should improve the [Schema Guideline] to extract the [Patients] according to [Cohort Selection] with [Target Features].

Output Format: Provide your response exactly as below, without additional commentary or text:

<think>

[Clearly and concisely explain your reasoning behind the classifications based on the given information in less than 5 sentences.]

</think>

<output>

<Add info> or <No info>

</output>

<Add info>

[Do not include information that is already in [Selected Schema] and [Schema Guideline]. Provide the gained information from [SQL Observation] in less than 6 sentences. This should be helpful to improve the [Schema Guideline] to extract the [Patients] according to [Cohort Selection] with [Target Features].]

</Add info>

[Original Schema]:{Original_Schema}

[Selected Schema]:{Selected_Schema}

[Target Features]:{Target_Features}

[Cohort Selection]:{Cohort_Selection}

[Previous Observation]:{Previous_Observation}

[SQL Observation]:{SQL_Observation}

G.1.4 Data Sufficiency Check

Data Sufficiency Check

You are an assistant to observe [SQL Observation] and find extra information to add to [Schema Guideline] to assist when generating SQL query for [Cohort Selection] patients with each of [Target Features].

Carefully review the following components:

- Original Schema: Includes tables, columns, and associated values before Schema Linking.

- Selected Schema: The schema and its guidelines chosen to extract patients according to [Cohort Selection] with specified features [Target Features].

- Target Features: Specific features required to extract for each patient.

- Cohort Selection: specifications for the configuration of patients to extract.

G.1.5 Update Schema Linking and Schema Guideline

Update Schema Linking and Schema Guideline

You are an assistant tasked with editing the [Schema Guideline] and [Schema] based on newly obtained [Additional Information]. Carefully review the following components:

- [Schema]: The original schema for [Target Features] and [Cohort Selection].

- [Schema Guideline]: The original schema guideline for [Target Features] and [Cohort Selection].

- [Additional Information]: New information gained from SQL Observation(s).

- [Target Features]: Specific features required to extract for each patient.

- [Cohort Selection]: specifications for the configuration of patients to extract.

Task:

- Make sure to update both [Schema Guideline] and [Schema] based on [Additional Information].
- Update the [Schema Guideline] and [Schema] based on [Additional Information] to support SQL query generation for extracting [Target Features] from the [Cohort Selection] patients.
- If [Additional Information] resolves previously unknown parts in [Schema Guideline], update them accordingly in [Schema Guideline].
- Provide the updated [Schema Guideline] no more than 15 sentences between <edited schema guideline> and </edited schema guideline>.
- Provide the updated [Schema] between <edited schema> and </edited schema> with the same format as the original [Schema] but with updated information such as column name, column type, column value (you can even add value examples), etc.
- If there is no need to update, provide the original [Schema Guideline] and [Schema].

Output Format: Provide your response exactly as below, without additional commentary or text:

<think>

[Explain your thought process clearly and concisely in no more than 5 sentences.]

</think>

<edited schema guideline>

[Edited Schema Guideline no more than 15 sentences]

</edited schema guideline>

<edited schema>

[Edited Schema]

</edited schema>

[Selected Schema]:{Selected_Schema}

[Schema Guideline]:{Schema_Guideline}

[Additional Information]:{Additional_Information}

[Target Features]:{Target_Features}

[Cohort Selection]:{Cohort_Selection}

G.1.6 SQL Generation

SQL Generation

Q: Using the provided [Schema] with tables and columns and [Schema Guideline], write a PostgreSQL query to extract patients according to [Cohort Selection] with specified features [Target Features]. Output is only the SQL query as plain text. Do not include code delimiters

Follow these steps:

1. Select appropriate foreign keys(columns) provided in [Relation Information] to connect identified tables.
2. If necessary, use selected foreign key to make "JOIN". Do not use any other columns.
3. Ensure that each column referenced in the SELECT clause is present in the table alias used.
4. Use [Requested Features] to follow the sequence and format of '(' in [Requested Features] to generate the SQL query.
5. If some values are not visually understandable due to mapping code, add 'CASE' and 'WHEN' to replace the values with understandable values.

6. When writing WHERE conditions involving categorical values (e.g., gender, status), Do not assume specific values.

7. Only use a specific value in WHERE clauses if it is explicitly observed in the schema or query result.

8. When applying multiple inclusion/exclusion criteria, ensure that logically dependent conditions are ordered correctly.

- Do not reorder or drop dependent conditions; maintain logical dependencies when translating natural language criteria into SQL.

9. For all float values in the SQL output, cast them to ::float in the SELECT clause. If rounding is applied, first cast to numeric for ROUND(..., n) to work, then cast the result back to ::float if a float output is desired.

10. In SQL WHERE clauses, string comparison is case-sensitive. Use LOWER(), UPPER(), or adjust collation if you need case-insensitive matching.

SQL generate rule:

- Ensure that the SQL query only applies numeric comparisons (such as BETWEEN) on values that are safely converted to integers, thereby preventing type conversion errors.

- Always extract the Patient ID as-is (without deduplication, filtering, or counting) for the first column, exactly as it appears in the database.

Output Format: Provide your response exactly as below, without additional commentary or text:

<think>

[Clearly and concisely explain your reasoning behind the sql generation based on the given information.]

</think>

<SQL query>

[Write a PostgreSQL query to extract requested features of patients according to [Cohort Selection] with [Requested Features]]

</SQL query>

Feedback Note: [Previous Failed SQL] and [Error Feedback] are failed SQL and error feedback. Carefully examine [Error Feedback] and avoid [Previous Failed SQL] to generate correct SQL.

[Cohort Selection]:{Cohort_Selection}

[Target Features]:{Target_Features}

[Selected Schema]:{Selected_Schema}

[Schema Guideline]:{Schema_Guideline}

[Previous Failed SQL]:{Previous_Failed_SQL}

[Error Feedback]:{Error_Feedback}

G.1.7 Error Feedback

Error Feedback

You are an assistant that classifies SQL execution errors.

Given:

- Failed SQL: The query that failed.
- Selected Schema: Schema used to generate the query.
- Target: Intended data to extract.
- Error Feedback: Database error message.

Task: Analyze the provided information and classify the error as one of the following:

<syntax error>: SQL syntax is incorrect (e.g., missing keywords, misplaced clauses, invalid syntax).

<wrong schema>: Schema-related issue (e.g., referencing non-existent tables or columns, incorrect schema usage).

Output Format: Provide your response exactly as below, without additional commentary or text:

<think>

[Explain your thought process clearly and concisely in less than 6 sentences, highlighting why you chose this classification and exactly what factors caused the error.]

</think>

<error class>

<syntax error> or <wrong schema>

</error class>

[Selected Schema]: {Selected_Schema}

[Schema Guideline]: {Schema_Guideline}

[Cohort Selection]: {Cohort_Selection}

[Target Features]: {Target_Features}

[Failed SQL]: {Failed_SQL}

[Error Feedback]: {Error_Feedback}

G.2 Prompts of CMA (Code Mapping Agent)

The following provides the detailed prompts used for CMA, as described in Section 3.3.

G.2.1 Schema Linking and Guideline Generation (Mapping Schema)

Schema Linking and Guideline Generation (for Mapping Schema)

Using [Database schema information], select all schema that are necessary to extract [Features].

- Please select exact table name(s) and column name(s) in [Database schema information].
- Follow the exact "Format" under [Notes] without any extra symbols, code delimiters.

[Notes]:

- Identify only definition schema (table(s), column(s), and 3 sample values for each column) related to [Feature].
- Exclude columns that have actual measurement values (vital sign or lab test).
- The columns of definition schema must include [Feature]'s information such as code, item number, name, abbreviation, etc.
- If identified definition table(s) have measurement unit information (not measurement value), get all the columns without actual measurement value information.
- After listing the schema for each feature, provide a [Schema Guideline] in a paragraph of no more than 10 sentences, explaining the details of the columns (such as type or how to interpret the values).

Output Format:

```
Mapping Table: dbname.Table_A , Column: Column_a,
Values: [value_1, value_2, value_3], Column:
Column_b,
Values: [value_1, value_2, value_3], Column:
Column_c
Mapping Table: dbname.Table_B , Column: Column_a,
Values: [value_1, value_2, value_3], Column:
Column_b,
Values: [value_1, value_2, value_3], Column:
Column_c
[Schema Guideline]: (paragraph of no more than 10
sentences)
```

[Schema Information]: {Schema_information}

[Curated database notes]: {Evaluation_Memo}

[Database Manual]: {Database_Manual}

[Feature]: {Feature_Selection}

G.2.2 Schema Linking and Guideline Generation (Feature Schema)

Schema Linking and Guideline Generation (for Feature Schema)

Using [Database schema information], select all schema that are necessary to extract [requested feature].

- Please select exact table name(s) and column name(s) in [Database schema information].
- Follow the exact "Format" under [Notes] without any extra symbols, code delimiters.

[Notes]:

- Select all schema (tables, columns, and 10 sample values for each column) related to extract [Feature], including definition table(s) and measurement table(s) of [Feature].
- The selected schema must include tables such as definition table(s) and measurement table(s) of [Feature].
- Provide a [Schema Guideline] in a paragraph of no more than 5 sentences, explaining the details of the columns (such as type or how to interpret the values).

Output Format:

<selected schema>

```
Table Name: dbname.Table_A , Column: Column_a,
Values: [value_1, value_2, value_3, value_4, ..., value_10]
Table Name: dbname.Table_B , Column: Column_1,
Values: [value_1, value_2, value_3, value_4, value_5, ...,
value_10]
```

</selected schema>

<schema guideline>

[Schema Guideline in a paragraph of no more than 5 sentences]

</schema guideline>

[Schema Information]: {Schema_information}

[Curated database notes]: {Evaluation_Memo}

[Database Manual]: {Database_Manual}

[Feature]: {Feature_Selection}

G.2.3 Feature Locating

Feature Locating

Classify whether the [Feature] name is literally present in any column name(s) of [Selected Schema]. If necessary, use [Schema Guideline] to help you classify whether the [Feature] name is literally present in any column name(s) of [Selected Schema].

If the [Feature] name is literally present in any column name(s) (e.g., [Feature]: 'chris', and [Selected Schema] has column names 'Destin', 'tom', 'CHRIS'), return it as SchemaName.TableName.ColumnName between <featurecolumn> and </featurecolumn>.

- Matching should be case-insensitive, space-insensitive, and symbol-insensitive. Reasonable abbreviations are also accepted.

- Do not match semantic or contextual similarity. Only match if [Feature] name is a literal substring of the column name after removing case, space, and symbol differences.
- If more than one column name is present in [Selected Schema], return all of them in <feature column> separated by || as SchemaName.TableName.ColumnName || SchemaName.TableName.ColumnName || ...

- Never match based on content or examples of values in the column.

If the [Feature] name is not literally present in any column name(s) (e.g., [Feature]: 'chris', and [Selected Schema] has column names 'name', 'tom', 'Andy'), output <feature column>None</feature column>.

- If the [Feature] name only matches semantically or through contextual similarity, but not literally, output <feature column>None</feature column>.

Output Format: Provide exactly:

<think>

[Explain your thought process clearly and concisely in no more than 5 sentences.]

</think>

<feature column>

[SchemaName.TableName.ColumnName if [Feature] name is literally present in any column name(s) from [Selected Schema], or None if not.]

</feature column>

G.2.4 Candidate Listing

Candidate Listing

Q: Using the provided Schema (tables, columns, values) and [Schema Guideline], generate a single PostgreSQL query to obtain columns 'unique feature identifier code (if exists)', 'feature name' and 'unit' from [Definition table].

To make a SQL query, follow these steps:

1. Identify tables that appear both in the [Feature Schema] and [Definition Schema]. Avoid using tables that are not in 'both' [Definition Schema] and [Feature Schema].

2. For identified table, ensure to obtain columns in the order of 'unique feature identifier code (if exists)', 'feature name' and 'unit'.

- Obtain 'unique feature identifier code' that represents feature types or items, but not row-level event-level IDs.

- Do NOT include the actual measurement value column.

3. If the table does not have a column about unit, look up the [Relation information] and [Feature Schema] to find any table that could provide the unit information via a foreign key relationship (e.g., measurement id, machine id, etc.). Then JOIN that table to retrieve the correct unit column.

4. Use consistent aliasing for each table (e.g., table AS alias) and ensure all aliases used in the SELECT clause are defined in the FROM clause.

5. Only use JOIN when necessary.

- Do not JOIN between each tables in [Definition Schema].

- Use JOIN only when there is a connection (foreign key) in [Relation Information].

6. Ensure the 'feature name' represents name of vital sign or lab test but not type or code number.

7. The order of the columns in the SELECT clause must be 'feature code number', 'feature name', and 'unit'.

Note for SQL formation:

1. Your final answer for each query must start from 'SELECT' (do not include any code fences or explanation).

2. Use DISTINCT to eliminate duplicate feature names. Return only one row per unique feature name.

3. When [Failed SQL] exists, carefully review the [Failed SQL] and [Error Feedback] to identify the cause of the failure and avoid the same mistake in the next SQL generation.

Output Format:

<think>

[Clearly and concisely explain your reasoning behind your SQL query generation.]

</think>

<SQL queries>

[SQL QUERY HERE]

</SQL queries>

G.2.5 Target and Candidates Matching Strategy

Since there can be a large number of candidates, the LLM internally filters out those with low similarity to the target feature during the initial **Target and Candidates Matching** step. In this way, only the most relevant candidates are presented, rather than displaying all candidates and their probabilities. User-defined threshold is applied in the second **Target and Candidates Matching** step to further filter candidates.

First Target and Candidates Matching

Compare each of the [Targeting Features] with each tuple from [Candidate Features] using your medical knowledge. Assign similarity probabilities within a range of 0 to 100 for each pair, ensuring the comparisons reflect the degree to which each Candidate Feature aligns with the specific Targeting Feature.

Only include [Candidate Features] tuple(s) with similarity probabilities that is equal or higher than the specified Similarity Threshold.

Formatting Requirements:

1. Targeting Features: Each result must begin with the name of the Targeting Feature, followed by a colon (:).

2. Candidate Features and Probabilities: After the colon, include a dictionary where:

- Each Candidate Feature is a key (tuple format).

- The assigned probability (0 to 100 integer only) reflects how strongly the Candidate Feature belongs to the same category or type as the Targeting Feature.

- Only unique candidate feature tuples should be included (i.e., do not repeat the same candidate feature multiple times).

3. Key-Value Separators: Use double-pipes || to separate key-value pairs inside the dictionary.

4. Separator: Use a semicolon (;) to separate results for each Targeting Feature.

5. No Additional Text: The output must strictly adhere to this format, and do not include code delimiters.

Example Input:

[Targeting Features]: Heart Rate

[Candidate Features]: [('C-reactive protein',), ('Pulse',), ('Serum Glucose',), ('SBP',)]

[Threshold]: 10

[Similarity Probabilities]:

('C-reactive protein',): 10|| ('Pulse',): 90

[Targeting Feature]:{Targeting_Feature}

[Candidate Features]:{Candidate_Features}

[Threshold]:{User_defined_threshold}

[Similarity Probabilities]:

Second Target and Candidates Matching

Compare the [Targeting Feature] with each tuple in [Synonyms] using medical knowledge.

Assign probabilities within a range of 0 to 100 for each pair, ensuring the comparisons reflect how strongly each Synonym belongs to the same category or type as the specific Targeting Feature.

Only include similarity probabilities that is equal or higher than the specified threshold.

Formatting Requirements:

1. Targeting Features: Each result must begin with the name of the Targeting Feature, followed by a colon (:).

2. Synonyms and Probabilities: After the colon, include a dictionary where:

- Each Synonym is a key (tuple format).

- The assigned probability (0 to 100) reflects how strongly the Synonym belongs to the same category or type as the Targeting Feature.

- Only unique synonym tuples should be included (i.e., do not repeat the same synonym multiple times).

3. Key-Value Separators: Use double-pipes (||) to separate key-value pairs inside the dictionary.

4. No Additional Text: The output must strictly adhere to this format, and do not include code delimiters.

Example 1:

[Targeting Features]: CRP

[Synonyms]: ('C-reactive protein',), ('Pulse',), ('Serum Glucose',), ('SBP',) Similarity Threshold: 1

[Similarity Probabilities]: CRP: ('C-reactive protein',): 99|| ('SBP',): 3

Example 2:

[Targeting Features]: Heart Rate

[Synonyms]: ('C-reactive protein',), ('Pulse',), ('Serum Glucose',), ('SBP',)

Similarity Threshold: 80

[Similarity Probabilities]: Heart Rate: ('Pulse',): 95

Example 3:

[Targeting Features]: SBP (mmHg)

[Synonyms]: ('Systolic Blood Pressure', 'mmHg'), ('Diastolic Blood Pressure', 'mmHg'), ('Heart Rate', 'bpm'), ('Serum Glucose', 'mg/dL') Similarity Threshold: 50

[Similarity Probabilities]: SBP: ('Systolic Blood Pressure', 'mmHg'): 98 || ('Diastolic Blood Pressure', 'mmHg'): 60

[Target Feature]:{Target_Feature}

[Candidate Features]:{Candidate_Features}

[Threshold]:{User_defined_threshold}

[Similarity Probabilities]:

for reproducibility and data-governance transparency. The controlled-backbone claude-3-5-sonnet-20240620 experiments were conducted before August 2025. At that time, we referred to the public PhysioNet guidance available from April 2023, which discussed responsible use of MIMIC data with online services and listed Anthropic Claude as a service whose prompts and responses were not used for model training by default and for which routine human review was not performed.⁵

PhysioNet subsequently issued updated guidance in September 2025 emphasizing that researchers are responsible for verifying zero data retention, no training use, and no human review when using cloud or API services, and strongly recommending locally deployed LLMs for MIMIC data.⁶ Consistent with this updated guidance, our current-backbone experiments use Qwen3.5-27B as a locally deployed open-weight model. In these runs, database contents and SQL observations remain within the local execution environment and are not sent to a third-party model API.

For auditability, we release prompts, generated SQL queries, selected mapping-code outputs, evaluation scripts, model identifiers, run dates, decoding settings, SQL-observation budgets, and retry limits, subject to the restriction that patient-level data and raw database contents are not redistributed.

H Appendix H. Data Governance and Reproducibility Details

All experiments use de-identified, credentialed-access PhysioNet datasets. All PhysioNet-hosted resources are cited with their dataset-specific records and the standard PhysioNet citation (Goldberger et al., 2000). PreCISE-EMR is released as a benchmarking codebase, not as a redistributed dataset: users must obtain the required credentials and execute the benchmark locally, and we do not release derived patient-level records.

We also document the model-use timeline

⁵<https://physionet.org/news/post/gpt-responsible-use/>

⁶<https://physionet.org/news/post/llm-responsible-use/>