EHRAgent: Code Empowers Large Language Models for Few-shot Complex Tabular Reasoning on Electronic Health Records

Wenqi Shi[1*], **Ran Xu**[2*], **Yuchen Zhuang**[1], **Yue Yu**[1], **Jieyu Zhang**[3], **Hang Wu**[1], **Yuanda Zhu**[1]
Joyce C. Ho[2], **Carl Yang**[2], **May D. Wang**[1,2]
Georgia Institute of Technology[1] Emory University[2] University of Washington[3]
{wqshi,yczhuang,yueyu,hangwu,yzhu94,maywang}@gatech.edu
{ran.xu,joyce.c.ho,j.carlyang}@emory.edu jieyuz2@cs.washington.edu

Abstract

Large language models (LLMs) have demonstrated exceptional capabilities in planning and tool utilization as autonomous agents, but few have been developed for medical problem-solving. We propose EHRAgent, an LLM agent empowered with a code interface, to autonomously generate and execute code for complex clinical tasks within electronic health records (EHRs). First, we formulate an EHR question-answering task into a tool-use planning process, efficiently decomposing a complicated task into a sequence of manageable actions. By integrating interactive coding and execution feedback, EHRAgent learns from error messages and improves the originally generated code through iterations. Furthermore, we enhance the LLM agent by incorporating long-term memory, which allows EHRAgent to effectively select and build upon the most relevant successful cases from past experiences. Experiments on three real-world multi-tabular EHR datasets show that EHRAgent outperforms the strongest baseline by up to 29.60%. EHRAgent leverages the emerging few-shot learning capabilities of LLMs, enabling autonomous code generation and execution to tackle complex clinical tasks.

1 Introduction

An electronic health record (EHR) is a digital version of a patient's medical history maintained by healthcare providers over time (Gunter & Terry, 2005). In clinical research and practice, clinicians actively interact with EHR systems to access and retrieve patient data, ranging from detailed individual-level records to comprehensive population-level insights (Cowie et al., 2017).

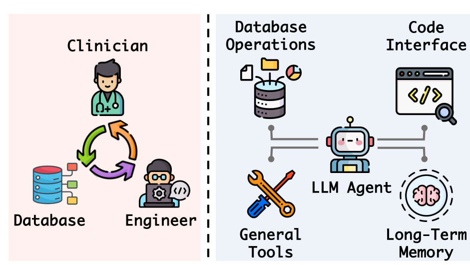


Figure 1: Simple and efficient interactions between clinicians and EHR systems with the assistance of LLM agents. Clinicians specify tasks in natural language, and the LLM agent autonomously generates and executes code to interact with EHRs (*right*) for answers. It eliminates the need for specialized expertise or extra effort from data engineers, which is typically required when dealing with EHRs in clinical settings (*left*).

Since most EHRs use pre-defined rule-based conversation systems (*e.g.*, Epic), clinicians may take additional training or seek help from data engineers to obtain information beyond rules (Mandel et al., 2016). Alternatively, an autonomous agent could facilitate clinicians to communicate with EHRs in natural languages, translating clinical questions into machine-interpretable queries (Lee et al., 2022), planning a sequence of actions, and ultimately delivering the final responses, which holds great potential to simplify workflows and reduce workloads for clinicians (Figure 1).

*Equal contribution.

Large language models (LLMs) (OpenAI, 2023; Anil et al., 2023) bring us one step closer to autonomous agents, with extensive knowledge and substantial instruction-following abilities from diverse corpora during pretraining. LLM-based autonomous agents have demonstrated remarkable capabilities in problem-solving, such as reasoning (Wei et al., 2022; Wang et al., 2023e; Zhou et al., 2023), planning (Yao et al., 2023b; Liu et al., 2023; Sun et al., 2023; Hao et al., 2023), and memorizing (Wang et al., 2023b). One particularly notable capability of LLM agents is *tool-usage* (Schick et al., 2023; Qin et al., 2023a), where they can utilize external tools (*e.g.*, calculators, APIs, *etc.*), interact with environments, and generate action plans with intermediate reasoning steps that can be executed sequentially towards a valid solution (Wu et al., 2023; Zhang et al., 2023).

Despite their success in general-domain tasks, LLMs have encountered unique but significant challenges when it comes to real-world clinical research and practice (Jiang et al., 2023; Yang et al., 2022; Moor et al., 2023), especially for EHRs that have complex structures and require additional information and expertise beyond their pre-trained data. First, given the constraints in both the volume and specificity of training data within the medical field, LLMs still struggle with medical reasoning due to insufficient knowledge and understanding of EHRs (Thapa & Adhikari, 2023). Second, EHRs are typically relational databases containing vast amounts of tables (*e.g.*, 26 tables in MIMIC-III (Johnson et al., 2016)) with heterogeneous patient data, including both administrative and clinical information. Moreover, unlike standardized questions (*e.g.*, multi-choice) found in medical licensing exams (Jin et al., 2021), real-world clinical tasks are highly diverse and complex (Lee et al., 2022). These questions often arise from the unique circumstances of individual patients or specific groups, necessitating multi-step or complicated operations.

To address these limitations, we propose `EHRAgent`, an autonomous LLM agent with external tools and code interface for improved multi-tabular reasoning across EHRs. We transform the EHR question-answering problem into a tool-use planning process - generating, executing, debugging, and optimizing a sequence of code-based actions. To overcome the lack of domain knowledge, we integrate additional information (*e.g.*, detailed descriptions of each table in EHRs) and clinical knowledge by instructing the LLM agent to retrieve the most relevant knowledge. We then establish an interactive coding mechanism, which involves a multi-turn dialogue between the code planner and executor, iteratively refining the generated code-based plan. Specifically, `EHRAgent` optimizes the execution plan by incorporating environment feedback and delving further into error information to enhance debugging proficiency. In addition, we take advantage of long-term memory to continuously maintain a set of successful cases and dynamically select the most relevant few-shot examples, in order to effectively learn from and improve upon past experiences.

We conducted extensive experiments on three widely used real-world EHR datasets, MIMIC-III (Johnson et al., 2016), eICU (Pollard et al., 2018), and TREQS (Wang et al., 2020), to validate the empirical effectiveness of `EHRAgent`, with a particular focus on challenging tasks that align with real-world application scenarios. In contrast to traditional supervised learning methods that require extensive training samples with fine-grained annotations (e.g., text-to-SQL (Lee et al., 2022)), `EHRAgent` demonstrates its efficiency by necessitating only four demonstrations. Our findings suggest that `EHRAgent` enables multi-tabular reasoning on EHRs by autonomous code generation and execution with environmental feedback. To the best of our knowledge, `EHRAgent` represents one of the first LLM agents for complex medical reasoning on EHRs with external tools and code interface.

Our main contributions are as follows:

• We propose `EHRAgent`, an LLM agent augmented with tools and medical knowledge, to solve multi-tabular reasoning derived from EHRs;

• Planning with a code interface, `EHRAgent` enables the LLM agent to formulate a clinical problem-solving process as an executable code plan of action sequences, along with a code executor;

• We introduce interactive coding between the LLM agent and code executor, iteratively refining plan generation and optimizing code execution by examining environment feedback in depth.

2 PRELIMINARIES

Problem Formulation. In this work, we focus on addressing health-related queries by leveraging information from structured EHRs. The reference EHR, denoted as $\mathcal{R} = \{R_0, R_1, \cdots\}$, comprises

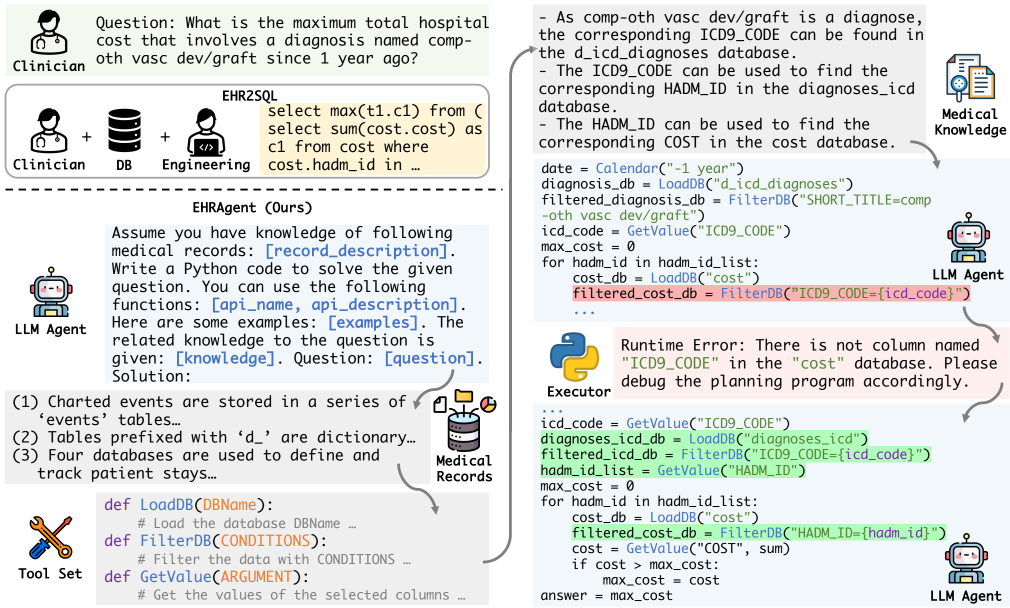


Figure 2: Overview of our proposed LLM agent, EHRAgent, for complex tabular reasoning tasks on EHRs. Given an input clinical question based on EHRs, EHRAgent initially incorporates relevant medical knowledge. Subsequently, EHRAgent decomposes the task and generates a plan (*i.e.*, code) based on EHR descriptions, tool function definitions, few-shot examples, and integrated medical knowledge. Upon execution, EHRAgent iteratively debugs the code following the environmental feedback and ultimately generates the final solution.

multiple tables, while $\mathcal{C} = \{C_0, C_1, \cdots\}$ corresponds to the column descriptions within $\mathcal{R}$. For each given query in natural language, denoted as q, our goal is to extract the final answer by utilizing the information within both $\mathcal{R}$ and $\mathcal{C}$.

LLM Agent Setup. We further formulate the planning process for LLMs as autonomous agents in EHR question answering. For initialization, the LLM agent is equipped with a set of pre-built tools $\mathcal{M} = \{M_0, M_1, \cdots\}$ to interact with and address queries derived from EHRs $\mathcal{R}$. Given an input query $q \in \mathcal{Q}$ from the task space $\mathcal{Q}$, the objective of the LLM agent is to design a T-step execution plan $P = (a_1, a_2, \cdots, a_T)$, with each action a_t selected from the tool set $a_t \in \mathcal{M}$. Specifically, we generate the action sequences (*i.e.*, plan) by prompting the LLM agent following a policy $p_q \sim \pi(a_1, \cdots, a_{T_q}|q; \mathcal{R}, \mathcal{M}) : \mathcal{Q} \times \mathcal{R} \times \mathcal{M} \to \Delta(\mathcal{M})^{T_q}$, where $\Delta(\cdot)$ is a probability simplex function. The final output is obtained by executing the entire plan $y \sim \rho(y|q, a_1, \cdots, a_{T_q})$, where ρ is a plan executor interacting with EHRs.

Planning with Code Interface. To mitigate ambiguities and misinterpretations in plan generation, an increasing number of LLM agents (Gao et al., 2023; Sun et al., 2023; Chen et al., 2023a) employ code prompts as planner interface instead of natural language prompts. The code interface enables LLM agents to formulate an executable code plan as action sequences, intuitively transforming natural language question-answering into iterative coding (Yang et al., 2023). Consequently, the planning policy $\pi(\cdot)$ turns into a code generation process, with a code execution as the executor $\rho(\cdot)$. We then track the outcome of each interaction back to the LLM agent, which can be either a successful execution result or an error message, to iteratively refine the generated code-based plan. This interactive process, a multi-turn dialogue between the planner and executor, takes advantage of the advanced reasoning capabilities of LLMs to optimize plan refinement and execution.

3 EHRAgent: LLMs as Medical Agents

In this section, we present EHRAgent (Figure 2), an LLM agent that enables multi-turn interactive coding to address multi-hop reasoning tasks on EHRs. EHRAgent comprises four key components: (1) **Medical Knowledge Integration:** EHRAgent summarizes the most important relevant information to facilitate a comprehensive understanding of EHRs. (2) **Interactive Coding with Execution Feedback:** `EHRAgent` harnesses LLMs as assistant agents in a multi-turn conversation with a code executor. (3) **Debugging via Error Tracing:** Rather than simply sending back information from the code executor, `EHRAgent` thoroughly analyzes error messages to identify the root causes through iterations until a final solution. (4) **Plan Refinement with Long-Term Memory:** Using long-term memory, `EHRAgent` selects the most relevant successful cases as demonstrations from past experiences for effective plan refinement. We summarize the workflow of `EHRAgent` in Algorithm 1.

3.1 Medical Knowledge Integration

We first incorporate medical knowledge into `EHRAgent` for a comprehensive understanding of EHRs within a limited context length. Given an EHR-based clinical question q and the reference EHRs $\mathcal{R} = \{R_0, R_1, \cdots\}$, the objective of knowledge integration is to generate descriptions of knowledge most relevant to q, thereby facilitating the identification and location of potential useful references within $\mathcal{R}$. For example, given a query related to '*Aspirin*', we expect LLMs to locate the drug '*Aspirin*' at the PRESCRIPTION table, under the *prescription_name* column in the EHR.

To achieve this, we initially maintain a thorough introduction $\mathcal{I}$ of all the reference EHRs, including overall data descriptions $\mathcal{D}$ and the detailed columnar descriptions $\mathcal{C}_i$ for each individual EHR R_i, expressed as $\mathcal{I} = [\mathcal{D}; \mathcal{C}_0; \mathcal{C}_1; \cdots]$. To further extract additional background knowledge essential for addressing the complex query q, we then distill key information from the detailed introduction $\mathcal{I}$. Specifically, we directly prompt LLMs to generate the relevant knowledge $B(q)$ based on demonstrations, denoted as $B(q) = \mathrm{LLM}([\mathcal{I}; q])$.

Algorithm 1: The procedure of `EHRAgent`.

Input: q: input question; $\mathcal{R}$: reference EHRs; $\mathcal{C}_i$: column description of EHR R_i; $\mathcal{D}$: descriptions of EHRs $\mathcal{R}$; T: the maximum number of steps; $\mathcal{T}$: definitions of tool function.

Initialize $t \leftarrow 0$, $C^{(0)}(q) \leftarrow \varnothing$, $O^{(0)}(q) \leftarrow \varnothing$

// Medical Knowledge Integration

$\mathcal{I} = [\mathcal{D}; \mathcal{C}_0; \mathcal{C}_1; \cdots]$

$B(q) = \mathrm{LLM}([\mathcal{I}; q])$

// Examples Retrieval from Long-Term Memory

$\mathcal{E}(q) = \arg\mathrm{TopK}_{\max}(\mathrm{sim}(q, q_i | q_i \in \mathcal{L}))$

// Plan Generation

$C^{(0)}(q) = \mathrm{LLM}([\mathcal{I}; \mathcal{T}; \mathcal{E}(q); q; B(q)])$

while $t < T$ & $\mathrm{TERMINATE} \notin O^{(t)}(q)$ **do**

  // Code Execution

  $O^{(t)}(q) = \mathrm{EXECUTE}(C^{(t)}(q))$

  // Debugging and Plan Modification

  $C^{(t+1)}(q) = \mathrm{LLM}(\mathrm{DEBUG}(O^{(t)}(q)))$

  $t \leftarrow t + 1$

Output: Final answer (solved) or error message (unsolved) from $O^{(t)}(q)$.

3.2 Interactive Coding with Execution

We then introduce interactive coding between the LLM agent (*i.e.*, code generator) and code executor to facilitate iterative plan refinement. `EHRAgent` integrates LLMs as an assistant agent with a code executor within a multi-turn conversation. The code executor retrieves and executes the generated code and then provides the execution results back to the LLM. Within the conversation, `EHRAgent` navigates the subsequent phase of the dialogue, where the LLM agent is expected to either (1) continue to iteratively adjust its original code in response to any errors encountered or (2) finally deliver a conclusive answer based on the successful execution outcomes.

LLM Agent. To generate accurate code snippets $C(q)$ as solution plans for the query q, we prompt the LLM agent with a combination of the EHR introduction $\mathcal{I}$, tool function definitions $\mathcal{T}$, a set of K-shot examples $E_1, \cdots, E_K$, the input query q, and the integrated medical knowledge relevant to the query $B(q)$:

$$C(q) = \mathrm{LLM}([\mathcal{I}; \mathcal{T}; E_1, \cdots, E_K; q; B(q)]). \tag{1}$$

Leveraging the AutoGen infrastructure (Wu et al., 2023) of automated multi-agent conversation, we develop the LLM agent to (1) generate code within a designated coding block as required, (2) modify the code according to the outcomes of its execution, and (3) insert a specific code "TERMINATE" at the end of its response to indicate the conclusion of the conversation.

Code Executor. The code executor automatically extracts the code from the LLM agent's output and executes it within the local environment: $O(q) = \text{EXECUTE}(C(q))$. After execution, it sends back the execution results to the LLM agent for potential plan refinement and further processing.

3.3 Error Tracing via Rubber Duck Debugging

Our empirical observations indicate that LLM agents tend to make slight modifications to the code snippets based on the error message without further debugging. In contrast, human programmers often delve deeper, identifying bugs or underlying causes by analyzing the code implementation against the error descriptions (Chen et al., 2023b). Inspired by this, we apply a 'rubber duck debugging' pipeline for plan refinement with the LLM agent. Specifically, we provide detailed trace feedback, including error type, message, and location, all parsed from the error information by the code executor. Subsequently, this error context is presented to a 'rubber duck' LLM, prompting it to generate the most probable causes of the error. The generated explanations are then fed back into the conversation flow, aiding in the debugging process. For the t-th interaction between the LLM agent and the code executor, the process is as follows:

$$\begin{aligned} O^{(t)}(q) &= \text{EXECUTE}(C^{(t)}(q)), \\ C^{(t+1)}(q) &= \text{LLM}(\text{DEBUG}(O^{(t)}(q))). \end{aligned} \tag{2}$$

The interaction ends either when a 'TERMINATE' signal appears in the generated messages or when t reaches a pre-defined threshold T.

3.4 Plan Refinement with Long-term Memory

Due to the vast volume of information within EHRs and the complexity of the clinical questions, there exists a conflict between limited input context length and the number of few-shot examples. Specifically, K-shot examples may not adequately cover the entire question types as well as the EHR information. To address this, we maintain a long-term memory $\mathcal{L}$ for storing past successful code snippets and reorganizing few-shot examples by retrieving the most relevant samples from $\mathcal{L}$. Consequently, the LLM agent can learn from and apply patterns observed in past successes to current queries. The selection of K-shot demonstrations $\mathcal{E}(q)$ is defined as follows:

$$\mathcal{E}(q) = \arg\text{TopK}_{\max}(\text{sim}(q, q_i | q_i \in \mathcal{L})), \tag{3}$$

where $\arg\text{TopK}\max(\cdot)$ identifies the indices of the top K elements with the highest values from $\mathcal{L}$, and $\text{sim}(\cdot,\cdot)$ calculates the similarity between two questions, employing negative Levenshtein distance as the similarity metric. Subsequent to this retrieval process, the newly acquired K-shot examples $\mathcal{E}(q)$ replace the originally predefined examples $E_1, \cdots, E_K$ in Eq. (1). This updated set of examples serves to reformulate the prompt, guiding the LLM agent in plan refinement:

$$C(q) = \text{LLM}([\mathcal{I}; \mathcal{T}; \mathcal{E}(q); q; B(q)]). \tag{4}$$

4 Experiments

4.1 Experiment Setup

Tasks and Datasets. We evaluate `EHRAgent` on three publicly available structured EHR datasets, MIMIC-III (Johnson et al., 2016), eICU (Pollard et al., 2018), and TREQS (Wang et al., 2020) for multi-hop question and answering on EHRs. These questions originate from real-world clinical needs and cover a wide range of tabular queries commonly posed within EHRs. During the data pre-processing stage, we create EHR question-answering pairs by considering text queries as questions and executing SQL commands in the database to automatically generate the corresponding ground-truth answers. Throughout this

Table 1: Dataset statistics.

Dataset	# Examples	# Table	# Row/Table	# Table/Q
MIMIC-III	580	17	81k	2.52
eICU	580	10	152k	1.74
TREQS	996	5	498k	1.48
Average	**718.7**	**10.7**	**243.7k**	**1.91**

Table 2: Main results of success rate (*i.e.*, SR.) and completion rate (*i.e.*, CR.) on MIMIC-III and eICU datasets. The complexity of questions increases from Level I (the simplest) to Level IV (the most difficult).

Dataset (→)	**MIMIC-III**						**eICU**					**TREQS**				
Complexity Level (→)	**I**	**II**	**III**	**IV**	**All**		**I**	**II**	**III**	**All**		**I**	**II**	**III**	**All**	
Methods (↓) /Metrics (→)	**SR.**				**SR.**	**CR.**	**SR.**			**SR.**	**CR.**	**SR.**			**SR.**	**CR.**
w/o Code Interface																
CoT (Wei et al., 2022)	29.33	12.88	3.08	2.11	9.58	38.23	26.73	33.00	8.33	27.34	65.65	11.22	9.15	0.00	9.84	54.02
Self-Consistency (Wang et al., 2023e)	33.33	16.56	4.62	1.05	10.17	40.34	27.11	34.67	6.25	31.72	70.69	12.60	11.16	0.00	11.45	57.83
Chameleon (Lu et al., 2023)	38.67	14.11	4.62	4.21	12.77	42.76	31.09	34.68	16.67	35.06	83.41	13.58	12.72	4.55	12.25	60.34
ReAct (Yao et al., 2023b)	34.67	12.27	3.85	2.11	10.38	25.92	27.82	34.24	15.38	33.33	73.68	33.86	26.12	9.09	29.22	78.31
Reflexion (Shinn et al., 2023)	41.05	19.31	12.57	11.96	19.48	57.07	38.08	33.33	15.38	36.72	80.00	35.04	29.91	9.09	31.53	80.02
w/ Code Interface																
LLM2SQL (Nan et al., 2023)	23.68	10.64	6.98	4.83	13.10	44.83	20.48	25.13	12.50	23.28	51.72	39.61	36.43	12.73	37.89	79.22
Self-Debugging (Chen et al., 2023b)	50.00	46.93	30.12	27.61	39.05	71.24	32.53	21.86	25.00	30.52	66.90	43.54	36.65	18.18	40.10	84.44
AutoGen (Wu et al., 2023)	36.00	28.13	15.33	11.11	22.49	61.47	42.77	40.70	18.75	40.69	86.21	46.65	19.42	0.00	33.13	85.38
`EHRAgent` (Ours)	**71.58**	**66.34**	**49.70**	**49.14**	**58.97**	**85.86**	**54.82**	**53.52**	**25.00**	**53.10**	**91.72**	**78.94**	**61.16**	**27.27**	**69.70**	**88.02**

process, we filter out samples containing unexecutable SQL commands or yielding empty results. In total, the final dataset involves 10.7 tables and 718.7 examples per dataset, with an average of 1.91 tables required to answer each question. Dataset statistics are available in Table 1. More details can be found in Appendix B.

Tool Sets. To enable LLMs in complex operations such as calculations and information retrieval, we integrate external tools in `EHRAgent` during the interaction with EHRs. Our toolkit can be easily expanded with natural language tool function definitions in a plug-and-play manner. Toolset details are available in Appendix C.

Baselines. We compare `EHRAgent` with the following LLMs-based planning, tool use, and coding baselines. We summarize and compare their key designs with `EHRAgent` in Table 4 in Appendix D.

◇ **CoT** (Wei et al., 2022): It enhances the complex reasoning capabilities of original LLMs by generating a series of intermediate reasoning steps.

◇ **Self-Consistency** (Wang et al., 2023e): It improves CoT by sampling diverse reasoning paths to replace the native greedy decoding and select the most consistent answer.

◇ **Chameleon** (Lu et al., 2023): It employs LLMs as controllers and integrates a set of plug-and-play modules, enabling enhanced reasoning and problem-solving across diverse tasks.

◇ **ReAct** (Yao et al., 2023b): It integrates reasoning with tool-use by guiding LLMs to generate intermediate verbal reasoning traces and tool commands.

◇ **Reflexion** (Shinn et al., 2023): It leverages verbal reinforcement to teach LLM-based agents to learn from linguistic feedback from past mistakes.

◇ **LLM2SQL** (Nan et al., 2023): It augments LLMs with a code interface to generate SQL queries for retrieving information from EHRs for question answering.

◇ **Self-Debugging** (Chen et al., 2023b): It teaches LLMs to debug by investigating execution results and explaining the generated code in natural language.

◇ **AutoGen** (Wu et al., 2023): It unifies LLM-based agent workflows as multi-agent conversations and uses the code interface to encode interactions between agents and environments.

Evaluation Protocol. Our primary evaluation metric is the *success rate*, quantifying the percentage of queries that the model successfully handles. Furthermore, we assess the *completion rate*, which represents the percentage of queries that the model is able to generate executable plans (even not yield correct results). We categorize input queries into various complexity levels (I-IV) based on the number of tables involved in solution generation (see details in Appendix B.2).

Implementation Details. We employ GPT-4 (OpenAI, 2023) as the base LLM model for all experiments. We set the temperature parameter (t) to 0 when making API calls to GPT-4 to eliminate randomness and set the pre-defined threshold (T) to 10. Due to the maximum length limitations of input context in baselines (*e.g.*, ReAct and Chameleon), we use the same initial four-shot demonstrations ($K = 4$) for all baselines and `EHRAgent` to ensure a fair comparison. Additional implementation details with prompt templates are available in Appendix E.

4.2 Main Results

Table 2 summarizes the experimental results of `EHRAgent` and baselines on multi-tabular reasoning within EHRs. From the results, we have the following observations:

◇ `EHRAgent` significantly outperforms all the baselines on all three datasets with a performance gain of 19.92%, 12.41%, and 29.60%, respectively. This indicates the efficacy of our key designs, namely interactive coding with environment feedback and domain knowledge injection, as they gradually refine the generated code and provide sufficient background knowledge during the planning process. Experimental results with additional base LLMs are available in Appendix F.1.

◇ *CoT*, *Self-Consistency*, and *Chameleon* all neglect environmental feedback and cannot adaptively refine their planning processes. Such deficiencies hinder their performance in EHR question-answering scenarios, as the success rate for these methods on three datasets is all below 40%.

◇ *ReAct* and *Reflexion* both consider environment feedback but are restricted to tool-generated error messages. Consequently, it potentially overlooks the overall planning process. Moreover, it lacks a code interface, which prevents it from efficient action planning, and results in lengthy context execution and lower completion rates.

◇ *LLM2SQL* leverages LLM to directly generate SQL queries for EHR question-answering tasks. However, the gain is rather limited, as the LLM still struggles to generate high-quality SQL codes for execution. Besides, the absence of a dedicated code debugging module further impedes its overall performance for this challenging task.

◇ *Self-Debugging* and *AutoGen* present a notable performance gain over other baselines, as they leverage code interfaces and consider the errors from the coding environment, leading to a large improvement in the completion rate. However, as they fail to model medical knowledge or identify inherent error patterns in the current code, the performance is still sub-optimal.

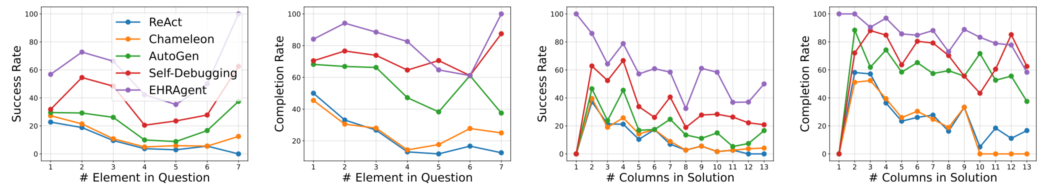
4.3 Quantitative Analysis

Ablation Studies. Our ablation studies on MIMIC-III (Table 3) demonstrate the effectiveness of all four components in `EHRAgent`. Interactive coding is the most significant contributor across all complexity levels, which highlights the importance of code generation in planning and environmental interaction for refinement. In addition, more challenging tasks benefits more from knowledge integration, indicating that comprehensive understanding of EHRs facilitates the complex multi-tabular reasoning in effective schema linking and reference (*e.g.*, tables, columns, and condition values) identification. Detailed analysis with additional results on eICU is available in Appendix F.2.

Table 3: Ablation studies on success rate (*i.e.*, SR.) and completion rate (*i.e.*, CR.) under different question complexity (I-IV) on MIMIC-III dataset.

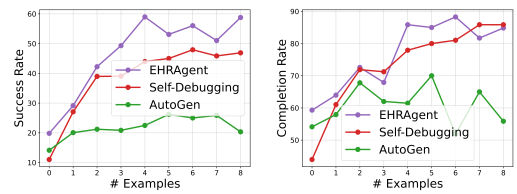
Complexity level	I	II	III	IV	All	
Metrics	SR.				SR.	CR.
EHRAgent	**71.58**	**66.34**	**49.70**	**49.14**	**58.97**	**85.86**
w/o medical knowledge	68.42	33.33	29.63	20.00	33.66	69.22
w/o interactive coding	45.33	23.90	20.97	13.33	24.55	62.14
w/o debugging	55.00	38.46	41.67	35.71	42.86	77.19
w/o long-term memory	65.96	54.46	37.13	42.74	51.73	83.42

Effect of Question Complexity. We take a closer look at the model performance by considering multi-dimensional measurements of question complexity, exhibited in Figure 3. Although the performances of both `EHRAgent` and the baselines generally decrease with an increase in task complexity (either quantified as more elements in queries or more columns in solutions), `EHRAgent` consistently outperforms all the baselines at various levels of difficulty. Appendix G.1 includes additional analysis on the effect of various question complexities.



(a) success rate (b) completion rate (c) success rate (d) completion rate

Figure 3: Success rate and completion rate under different question complexity, measured by the number of elements (*i.e.*, slots) in each question and the number of columns involved in each solution.



(a) success rate (b) completion rate

Figure 4: Success rate and completion rate under different numbers of demonstrations.

Sample Efficiency. Figure 4 illustrates the model performance *w.r.t.* number of demonstrations for `EHRAgent` and the two strongest baselines, AutoGen and Self-Debugging. Compared to supervised learning (*e.g.*, text-to-SQL (Wang et al., 2020; Raghavan et al., 2021; Lee et al., 2022)) that requires extensive training on over 10K samples with detailed annotations (*e.g.*, SQL code), LLM agents enable complex tabular reasoning using a few demonstrations only. One interesting finding is that as the number of examples increases, both the success and completion rate of AutoGen tend to decrease, mainly due to the context limitation of LLMs. On the contrary, the performance of `EHRAgent` remains stable with more demonstrations, which may benefit from its integration of a 'rubber duck' debugging module and the adaptive mechanism for selecting the most relevant demonstrations.

4.4 Error Analysis

Figure 5 presents a summary of error types identified in the solution generation process of `EHRAgent` based on the MIMIC-III dataset, as determined through manual examinations and analysis. The majority of errors occur because the LLM agent consistently fails to identify the root cause of these errors within T-step trails, resulting in plans that are either incomplete or inexcusable. Additional analysis of each error type is available in Appendix G.2.

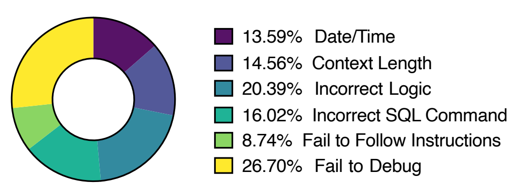


Figure 5: Percentage of mistake examples in different categories on MIMIC-III dataset.

4.5 Case Study

Figure 6 showcases a comparative case study of the interactive coding process between AutoGen and `EHRAgent` for the same given query. When executed with error feedback, AutoGen directly sends back the original error messages, making slight modifications (*e.g.*, changing the surface string of the arguments) without reasoning the root cause of the error. In contrast, `EHRAgent` can identify the underlying causes of the errors through interactive coding and debugging processes. It successfully discovers the underlying error causes (taking into account case sensitivity), facilitating accurate code refinement. Extensive case studies showcasing the effectiveness of different components in `EHRAgent` are available in Appendix H.

5 Related Work

Augmenting LLMs with External Tools. LLMs have rapidly evolved from text generators into core computational engines of autonomous agents, with advanced planning and tool-use capabilities (Schick et al., 2023; Shen et al., 2023; Wang et al., 2023d; Yuan et al., 2024a;b). LLM agents

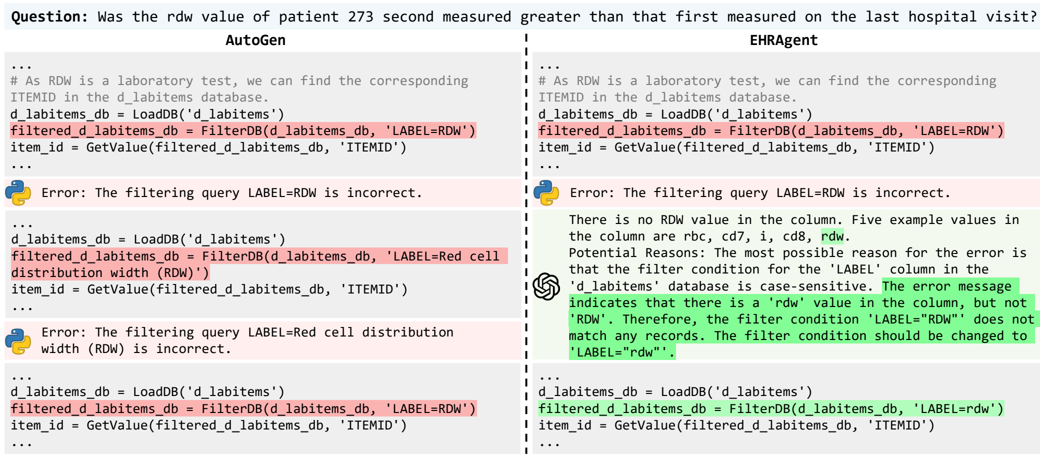


Figure 6: A case study of the interactive coding process between AutoGen (*left*) and EHRAgent (*right*), where EHRAgent delves deeper into environmental feedback to achieve plan refinement.

equip LLMs with planning capabilities (Wei et al., 2022; Yao et al., 2023a) to decompose a large and hard task into multiple smaller and simpler steps for efficiently navigating complex real-world scenarios. By integrating with external tools, LLM agents access external APIs for additional knowledge beyond training data (Shen et al., 2023; Huang et al., 2022; Sun et al., 2023; Lu et al., 2023; Patil et al., 2023; Qin et al., 2023b; Li et al., 2023), such as real-time information (Nakano et al., 2022; Parisi et al., 2022), computational capability (Schick et al., 2023), and coding proficiency (Wu et al., 2023; Zhang et al., 2023; Gao et al., 2023; Chen et al., 2023a; Nan et al., 2023). The disconnection between plan generation and execution, however, prevents LLM agents from effectively and efficiently preventing error propagation and learning from environmental feedback (Yao et al., 2023b; Shinn et al., 2023; Yang et al., 2023). To this end, we leverage interactive coding to learn from dynamic interactions between the planner and executor, iteratively refining generated code by incorporating insights from error messages. Furthermore, EHRAgent extends beyond the limitation of short-term memory obtained from in-context learning, leveraging long-term memory (Wang et al., 2023b; Chen et al., 2023a) by rapid retrieval of highly relevant and successful experiences accumulated over time.

LLM Agents for Scientific Discovery. Augmenting LLMs with domain-specific tools, LLM agents have demonstrated capabilities of autonomous design, planning, and execution in accelerating scientific discovery (Wang et al., 2023a;c; Xi et al., 2023; Zhao et al., 2023), including organic synthesis (Bran et al., 2023), material design (Boiko et al., 2023), and gene prioritization (Jin et al., 2023). In the medical field, MedAgents (Tang et al., 2023), the first multi-agent collaboration framework in medical domain, leverages role-playing LLM-based agents in a task-oriented multi-round discussion for multi-choice questions in medical entrance examinations. Similarly, Abbasian et al. (2023) develop a conversational agent to enhance LLMs using Langchain tools for general medical question and answering tasks. Different from existing LLM agents in medical and scientific domains, EHRAgent integrates LLMs with interactive code interface, targeting complex tabular tasks derived from real-world EHRs through autonomous code generation and execution.

6 Conclusion

In this study, we developed EHRAgent, an LLM agent equipped with an interactive code interface for multi-tabular reasoning on real-world EHRs. By leveraging the emergent few-shot learning capabilities of LLMs, EHRAgent enables autonomous code generation and execution to address complicated clinical tasks, including database operations on EHRs with minimal demonstrations. Furthermore, we improve EHRAgent by interactive coding with execution feedback, along with a long-term memory mechanism, thereby effectively facilitating plan optimization for multi-step problem-solving. Our experiments on real-world EHR datasets demonstrate the advantages of EHRAgent over baseline LLM agents in autonomous coding and improved medical reasoning. EHRAgent holds considerable potential for positive social impact in a wide range of clinical tasks and applications, including but not limited to patient cohort definition, clinical trial recruitment, case review selection, and treatment decision-making support.

REFERENCES

Mahyar Abbasian, Iman Azimi, Amir M. Rahmani, and Ramesh Jain. Conversational health agents: A personalized llm-powered agent framework, 2023.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report, 2023.

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023. URL https://www.nature.com/articles/s41586-023-06792-0.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew White, and Philippe Schwaller. Augmenting large language models with chemistry tools. In *NeurIPS 2023 AI for Science Workshop*, 2023. URL https://openreview.net/forum?id=wdGIL6lx3l.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023a. ISSN 2835-8856. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug, 2023b.

Martin R Cowie, Juuso I Blomster, Lesley H Curtis, Sylvie Duclaux, Ian Ford, Fleur Fritz, Samantha Goldman, Salim Janmohamed, Jörg Kreuzer, Mark Leenay, et al. Electronic health records to facilitate clinical research. *Clinical Research in Cardiology*, 106:1–9, 2017. URL https://link.springer.com/article/10.1007/s00392-016-1025-6.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Tracy D Gunter and Nicolas P Terry. The emergence of national electronic health record architectures in the united states and australia: models, costs, and questions. *Journal of medical Internet research*, 7(1):e383, 2005. URL https://www.jmir.org/2005/1/e3/.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.507. URL https://aclanthology.org/2023.emnlp-main.507.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pp. 9118–9147. PMLR, 2022. URL https://proceedings.mlr.press/v162/huang22a.html.

Lavender Yao Jiang, Xujin Chris Liu, Nima Pour Nejatian, Mustafa Nasir-Moin, Duo Wang, Anas Abidin, Kevin Eaton, Howard Antony Riina, Ilya Laufer, Paawan Punjabi, et al. Health system-scale language models are all-purpose prediction engines. *Nature*, pp. 1–6, 2023. URL https://www.nature.com/articles/s41586-023-06160-y.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021. URL https://www.mdpi.com/2076-3417/11/14/6421.

Qiao Jin, Yifan Yang, Qingyu Chen, and Zhiyong Lu. Genegpt: Augmenting large language models with domain tools for improved access to biomedical information, 2023.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016. URL `https://www.nature.com/articles/sdata201635`.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. EHRSQL: A practical text-to-SQL benchmark for electronic health records. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL `https://openreview.net/forum?id=B2W8Vy0rarw`.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=3IyL2XWDkG`.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+p: Empowering large language models with optimal planning proficiency, 2023.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=HtqnVSCj3q`.

Joshua C Mandel, David A Kreda, Kenneth D Mandl, Isaac S Kohane, and Rachel B Ramoni. Smart on fhir: a standards-based, interoperable apps platform for electronic health records. *Journal of the American Medical Informatics Association*, 23(5):899–908, 2016. URL `https://academic.oup.com/jamia/article/23/5/899/2379865`.

Michael Moor, Oishi Banerjee, Zahra Shakeri Hossein Abad, Harlan M Krumholz, Jure Leskovec, Eric J Topol, and Pranav Rajpurkar. Foundation models for generalist medical artificial intelligence. *Nature*, 616(7956):259–265, 2023. URL `https://www.nature.com/articles/s41586-023-05881-4`.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.0933*, 2022.

Linyong Nan, Ellen Zhang, Weijin Zou, Yilun Zhao, Wenfei Zhou, and Arman Cohan. On evaluating the integration of reasoning and action in llm agents with database question answering, 2023.

OpenAI. Gpt-4 technical report. *arXiv*, 2023.

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models, 2022.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023.

Tom J. Pollard, Alistair E. W. Johnson, Jesse D. Raffa, Leo A. Celi, Roger G. Mark, and Omar Badawi. The eICU collaborative research database, a freely available multi-center database for critical care research. *Scientific Data*, 5(1):180178, 2018. ISSN 2052-4463. URL `https://www.nature.com/articles/sdata2018178`.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. Tool learning with foundation models, 2023a.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023b.

Preethi Raghavan, Jennifer J Liang, Diwakar Mahajan, Rachita Chandra, and Peter Szolovits. emrkbqa: A clinical knowledge-base question answering dataset. In *Proceedings of the 20th Workshop on Biomedical Language Processing*, pp. 64–73, 2021. URL `https://aclanthology.org/2021.bionlp-1.7/`.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI tasks with chatGPT and its friends in hugging face. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=yHdTscY6Ci.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=rnKgbKmelt.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning, 2023.

Surendrabikram Thapa and Surabhi Adhikari. Chatgpt, bard, and large language models for biomedical research: opportunities and pitfalls. *Annals of Biomedical Engineering*, 51(12):2647–2651, 2023. URL https://link.springer.com/article/10.1007/s10439-023-03284-0.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents, 2023a.

Ping Wang, Tian Shi, and Chandan K Reddy. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pp. 350–361, 2020. URL https://dl.acm.org/doi/abs/10.1145/3366423.3380120.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL https://openreview.net/forum?id=BryMFPQ4L6.

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R. Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. Scibench: Evaluating college-level scientific problem-solving abilities of large language models, 2023c.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback, 2023d.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023e. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey, 2023.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback, 2023.

Xi Yang, Aokun Chen, Nima PourNejatian, Hoo Chang Shin, Kaleb E Smith, Christopher Parisien, Colin Compas, Cheryl Martin, Anthony B Costa, Mona G Flores, et al. A large language model for electronic health records. *NPJ Digital Medicine*, 5(1):194, 2022. URL `https://www.nature.com/articles/s41746-022-00742-2`.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL `https://openreview.net/forum?id=5Xc1ecxO1h`.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL `https://openreview.net/forum?id=WE_vluYUL-X`.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R Fung, Hao Peng, and Heng Ji. CRAFT: Customizing LLMs by creating and retrieving from specialized toolsets. In *The Twelfth International Conference on Learning Representations*, 2024a. URL `https://openreview.net/forum?id=G0vdDSt9XM`.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. Easytool: Enhancing llm-based agents with concise tool instruction, 2024b.

Jieyu Zhang, Ranjay Krishna, Ahmed H. Awadallah, and Chi Wang. Ecoassistant: Using llm assistant more affordably and accurately, 2023.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=WZH7099tgfM`.

A Limitation

One potential limitation is that while the framework of our proposed `EHRAgent` is broadly applicable to various scenarios, it currently relies on code generation for tool usage and problem-solving. Furthermore, the adaptation and generalization of `EHRAgent` in low-resource languages is constrained by the availability of relevant resources and training data. Additionally, given the demands for privacy, safety, and ethical considerations in real-world clinical settings, our goal is to further advance `EHRAgent` by mitigating biases and addressing ethical implications, thereby contributing to the development of responsible artificial intelligence for healthcare and medicine.

B Dataset Details

B.1 Task Details

We evaluate `EHRAgent` on two text-to-SQL medical question answering (QA) benchmarks (Lee et al., 2022), EHRSQL[1] and TREQS[2], built upon structured EHRs from MIMIC-III and eICU.

[1] `https://github.com/glee4810/EHRSQL`
[2] `https://github.com/wangpinggl/TREQS`

EHRSQL serves as a text-to-SQL benchmark for assessing the performance of medical QA models, specifically focusing on generating SQL queries for addressing a wide range of real-world questions gathered from over 200 hospital staff. Questions within EHRSQL, ranging from simple data retrieval to complex operations such as calculations, reflect the *diverse* and *complex* clinical tasks encountered by front-line healthcare professionals.

B.2 QUESTION COMPLEXITY LEVEL

We categorize input queries into various complexity levels (levels I-IV for MIMIC-III and levels I-III for eICU and TREQS) based on the number of tables involved in solution generation. For example, given the question 'How many patients were given temporary tracheostomy?', the complexity level is categorized as II, indicating that we need to extract information from two tables (admission and procedure) to generate the solution. Furthermore, we also conduct a performance analysis (see Figure 3) based on additional evaluation metrics related to question complexity, including (1) the number of elements (*i.e.*, slots) in each question and (2) the number of columns involved in each solution. Specifically, elements refer to the slots within each template that can be populated with pre-defined values or database records.

B.3 MIMIC-III

MIMIC-III (Johnson et al., 2016)[3] covers 38,597 patients and 49,785 hospital admissions information in critical care units at the Beth Israel Deaconess Medical Center ranging from 2001 to 2012. It includes deidentified administrative information such as demographics and highly granular clinical information, including vital signs, laboratory results, procedures, medications, caregiver notes, imaging reports, and mortality.

B.4 EICU

Similar to MIMIC-III, eICU (Pollard et al., 2018)[4] includes over 200,000 admissions from multiple critical care units across the United States in 2014 and 2015. It contains deidentified administrative information following the US Health Insurance Portability and Accountability Act (HIPAA) standard and structured clinical data, including vital signs, laboratory measurements, medications, treatment plans, admission diagnoses, and medical histories.

B.5 TREQS

TREQS (Wang et al., 2020) is a healthcare question and answering dataset that is built upon the MIMIC-III (Johnson et al., 2016) dataset. In TREQS, questions are generated automatically using pre-defined templates with the text-to-SQL task. Compared to the MIMIC-III dataset within the EHRSQL (Lee et al., 2022) benchmark, TREQS has a narrower focus in terms of the types of questions and the complexity of SQL queries. Specifically, it is restricted to only five tables but includes a larger number of records (498k) within each table.

C TOOL SET DETAILS

To obtain relevant information from EHRs and enhance the problem-solving capabilities of LLM-based agents, we augment LLMs with the following tools:

◇ **Database Loader** loads a specific table from the database.

◇ **Data Filter** applies specific filtering condition to the selected table. These conditions are defined by a column name and a relational operator. The relational operator may take the form of a comparison (e.g., "<" or ">") with a specific value, either with the column's values or the count of values grouped by another column. Alternatively, it could be operations such as identifying the minimum or maximum values within the column.

[3] `https://physionet.org/content/mimiciii/1.4/`
[4] `https://physionet.org/content/eicu-crd/2.0/`

◇ **Get Value** retrieves either all the values within a specific column or performs basic operations on all the values, including calculations for the mean, maximum, minimum, sum, and count.

◇ **Calculator** calculates the results from input strings. We leverage the WolframAlpha API portal[5], which can handle both straightforward calculations such as addition, subtraction, and multiplication and more complex operations like averaging and identifying maximum values.

◇ **Date Calculator** calculates the target date based on the input date and the provided time interval information.

◇ **SQL Interpreter** interprets and executes SQL code written by LLMs.

D Comparison of Baselines

We compare baselines and `EHRAgent` on the inclusion of different components in Table 4.

Table 4: Comparison of baselines and `EHRAgent` on the inclusion of different components.

Baselines	Tool Use	Code Interface	Environment Feedback	Debugging	Error Exploration	Medical Knowledge	Long-term Memory
w/o Code Interface							
CoT (Wei et al., 2022)	✓	✗	✗	✗	✗	✗	✗
Self-Consistency (Wang et al., 2023e)	✓	✗	✗	✗	✗	✗	✗
Chameleon (Lu et al., 2023)	✓	✗	✗	✗	✗	✗	✗
ReAct (Yao et al., 2023b)	✓	✗	✓	✗	✗	✗	✗
Reflexion (Shinn et al., 2023)	✓	✗	✓	✓	✗	✗	✗
w/ Code Interface							
LLM2SQL (Nan et al., 2023)	✗	✓	✗	✗	✗	✗	✗
Self-Debugging (Chen et al., 2023b)	✗	✓	✓	✓	✗	✗	✗
AutoGen (Wu et al., 2023)	✓	✓	✓	✓	✗	✗	✗
`EHRAgent` (Ours)	✓	✓	✓	✓	✓	✓	✓

E Additional Implementation Details

E.1 Hardware and Software Details

All experiments are conducted on CPU: Intel(R) Core(TM) i7-5930K CPU @ 3.50GHz and GPU: NVIDIA GeForce RTX A5000 GPUs, using Python 3.9 and AutoGen 0.2.0[6].

E.2 Code Generation Details

Given that the majority of LLMs have been pre-trained on Python code snippets (Gao et al., 2023), and Python's inherent modularity aligns well with tool functions, we choose Python 3.9 as the primary language for interaction coding between the LLM agent and the code executor.

E.3 Prompt Details

In the subsequent subsections, we detail the prompt templates employed in `EHRAgent`. The complete version of the prompts is available at our code repository due to space limitations.

◇ **Prompt for Code Generation.** We first present the prompt template for `EHRAgent` in code generation as follows:

<LLM_Agent> Prompt

```
Assume you have knowledge of several tables:
{OVERALL_EHR_DESCRIPTIONS}
Write a python code to solve the given question.
    You can use the following functions:
```

[5] `https://products.wolframalpha.com/api`
[6] `https://github.com/microsoft/autogen`

```
{TOOL_DEFINITIONS}
Use the variable 'answer' to store the answer
    of the code. Here are some examples:
{4-SHOT_EXAMPLES}
(END OF EXAMPLES)
Knowledge:
{KNOWLEDGE}
Question: {QUESTION}
Solution:
```

◇ **Prompt for Knowledge Integration.** We then present the prompt template for knowledge integration in `EHRAgent` as follows:

<Medical_Knowledge> Prompt

```
Read the following data descriptions, generate
    the background knowledge as the context
    information that could be helpful for
    answering the question.
{OVERALL_EHR_DESCRIPTIONS}
For different tables, they contain the
    following information:
{COLUMNAR_DESCRIPTIONS}

{4-SHOT_EXAMPLES}

Question: {QUESTION}
Knowledge:
```

◇ **Prompt for 'Rubber Duck' Debugging.** The prompt template used for debugging module in `EHRAgent` is shown as follows:

<Error_Exploration> Prompt

```
Given a question:
{QUESTION}
The user has written code with the following
    functions:
{TOOL_DEFINITIONS}

The code is as follows:
{CODE}

The execution result is:
{ERROR_INFO}

Please check the code and point out the most
    possible reason to the error.
```

◇ **Prompt for Few-Shot Examples.** The prompt template used for few-shot examples in `EHRAgent` is shown as follows:

<Few_Shot_Examples> Prompt

```
Question: {QUESTION_I}
Knowledge:
{KNOWLEDGE_I}
Solution: {CODE_I}

Question: {QUESTION_II}
Knowledge:
{KNOWLEDGE_II}
Solution: {CODE_II}
```

```
Question: {QUESTION_III}
Knowledge:
{KNOWLEDGE_III}
Solution: {CODE_III}

Question: {QUESTION_IV}
Knowledge:
{KNOWLEDGE_IV}
Solution: {CODE_IV}
```

F ADDITIONAL EXPERIMENTAL RESULTS

F.1 EFFECT OF BASE LLMS

Table 5 presents a summary of the experimental results obtained from EHRAgent and all baselines using a different base LLM, GPT-3.5-turbo. The results clearly demonstrate that EHRAgent continues to outperform all the baselines, achieving a performance gain of 6.72%. This highlights the ability of EHRAgent to generalize across different base LLMs as backbone models. When comparing the experiments conducted with GPT-4 (Table 2), it is evident that the performance of both the baselines and EHRAgent decreases. This can primarily be attributed to the weaker capabilities of instruction-following and reasoning in GPT-3.5-turbo.

Table 5: Experimental results of success rate (*i.e.*, SR.) and completion rate (*i.e.*, CR.) on MIMIC-III based on GPT-3.5-turbo as the base LLM. The complexity of questions increases from Level I (the simplest) to Level IV (the most difficult).

Dataset (→)	**MIMIC-III**					
Complexity Level (→)	**I**	**II**	**III**	**IV**	**All**	
Methods (↓) /Metrics (→)	**SR.**				**SR.**	**CR.**
w/o Code Interface						
CoT (Wei et al., 2022)	23.16	10.40	2.99	1.71	8.62	41.55
Self-Consistency (Wang et al., 2023e)	25.26	11.88	4.19	2.56	10.52	47.59
Chameleon (Lu et al., 2023)	27.37	11.88	3.59	2.56	11.21	47.59
ReAct (Yao et al., 2023b)	26.32	10.89	3.59	3.42	9.66	61.21
Reflexion (Shinn et al., 2023)	30.53	12.38	9.58	8.55	13.28	66.72
w/ Code Interface						
LLM2SQL (Nan et al., 2023)	21.05	15.84	4.19	2.56	10.69	59.49
Self-Debugging (Chen et al., 2023b)	36.84	33.66	22.75	16.24	27.59	72.93
AutoGen (Wu et al., 2023)	28.42	25.74	13.17	10.26	19.48	52.42
EHRAgent (Ours)	**43.16**	**42.57**	**29.94**	**18.80**	**34.31**	**78.80**

F.2 ADDITIONAL ABLATION STUDIES

We conduct additional ablation studies to evaluate the effectiveness of each module in EHRAgent on eICU in Table 6 and obtain consistent results. From the results from both MIMIC-III and eICU, we observe that all four components contribute significantly to the performance gain.

◇ **Medical Knowledge Integration.** Out of all the components, the medical knowledge injection module mainly exhibits its benefits in challenging tasks. These tasks often involve more tables and require a deeper understanding of domain knowledge to associate items with their corresponding tables.

◇ **Interactive Coding.** The interactive coding interface is the most significant contributor to the performance gain across all complexity levels. This verifies the importance of utilizing the code interface for planning instead of natural languages, which enables the model to avoid overly complex contexts and thus leads to a substantial increase in the completion rate. Additionally, the code interface also allows the debugging module to refine the planning with execution feedback, improving the efficacy of the planning process.

◇ **Debugging Module.** The 'rubber duck' debugging module enhances the performance by guiding the LLM agent to figure out the underlying reasons for the error messages. This enables `EHRAgent` to address the intrinsic error that occurs in the original reasoning steps.

◇ **Long-term Memory.** Following the reinforcement learning setting (Sun et al., 2023; Shinn et al., 2023), the long-term memory mechanism improves performance by justifying the necessity of selecting the most relevant demonstrations for planning. In order to simulate the scenario where the ground truth annotations (*i.e.*, rewards) are unavailable, we further evaluate the effectiveness of the long-term memory on the completed cases in Table 7, regardless of whether they are successful or not. The results indicate that the inclusion of long-term memory with completed cases increases the completion rate but tends to reduce the success rate across most difficulty levels, as some incorrect cases might be included as the few-shot demonstrations. Nonetheless, it still outperforms the performance without long-term memory, confirming the effectiveness of the memory mechanism.

Table 6: Additional ablation studies on success rate (*i.e.*, SR.) and completion rate (*i.e.*, CR.) under different question complexity (I-III) on eICU dataset.

Complexity level	I	II	III	All	
Metrics		SR.		SR.	CR.
EHRAgent	**54.82**	**53.52**	**25.00**	**53.10**	**91.72**
w/o medical knowledge	36.75	28.39	6.25	30.17	47.24
w/o interactive coding	46.39	44.97	6.25	44.31	65.34
w/o debugging	50.60	46.98	12.50	47.07	70.86
w/o long-term memory	52.41	44.22	18.75	45.69	78.97

Table 7: Comparison on long-term memory (*i.e.*, LTM) design under different question complexity (I-IV) on MIMIC-III dataset.

Complexity level	I	II	III	IV	All	
Metrics			SR.		SR.	CR.
EHRAgent (LTM w/ Success)	71.58	**66.34**	**49.70**	**49.14**	**58.97**	85.86
LTM w/ Completion	**76.84**	60.89	41.92	34.48	53.24	**90.05**
w/o LTM	65.96	54.46	37.13	42.74	51.73	83.42

G Additional Empirical Analysis

G.1 Additional Question Complexity Analysis

We further analyze the model performance by considering various measures of question complexity based on the number of elements in questions, and the number of columns involved in solutions, as shown in Figure 3. Incorporating more elements requires the model to either perform calculations or utilize domain knowledge to establish connections between elements and specific columns. Similarly, involving more columns also presents a challenge for the model in accurately locating and associating the relevant columns. We notice that both `EHRAgent` and baselines generally exhibit lower performance on more challenging tasks[7]. Notably, our model consistently outperforms all the baseline models across all levels of difficulty. Specifically, for those questions with more than 10 columns, the completion rate of those open-loop baselines is very low (less than 20%), whereas `EHRAgent` can still correctly answer around 50% of queries, indicating the robustness of `EHRAgent` in handling complex queries with multiple elements.

[7]Exceptions may exist when considering questions of seven elements in Figures 3(a) and 3(b), as it comprises only eight samples and may not be as representative.

G.2 Additional Error Analysis

We conducted a manual examination to analyze all incorrect cases generated by `EHRAgent` in MIMIC-III. Figure 5 illustrates the percentage of each type of error frequently encountered during solution generation:

◇ **Date/Time.** When addressing queries related to dates and times, it is important for the LLM agent to use the 'Calendar' tool, which bases its calculations on the system time of the database. This approach is typically reliable, but there are situations where the agent defaults to calculating dates based on real-world time. Such instances may lead to potential inaccuracies.

◇ **Context Length.** This type of error occurs when the input queries or dialog histories are excessively long, exceeding the context length limit.

◇ **Incorrect Logic.** When solving multi-hop reasoning questions across multiple databases, the LLM agent may generate executable plans that contain logical errors in the intermediate reasoning steps. For instance, in computing the total cost of a hospital visit, the LLM agent might erroneously generate a plan that filters the database using `patient_id` instead of the correct `admission_id`.

◇ **Incorrect SQL Command.** This error type arises when the LLM agent attempts to integrate the `SQLInterpreter` into a Python-based plan to derive intermediate results. Typically, incorrect SQL commands result in empty responses from `SQLInterpreter`, leading to the failure of subsequent parts of the plan.

◇ **Fail to Follow Instructions.** The LLM agent often fails to follow the instructions provided in the initial prompt or during the interactive debugging process.

◇ **Fail to Debug.** Despite undertaking all T-step trials, the LLM agent consistently fails to identify the root cause of errors, resulting in plans that are either incomplete or inexcusable.

H Additional Case Studies

We present additional case studies to showcase the effectiveness of each module in `EHRAgent`, including medical knowledge integration (Figure 7), code interface (Figures 8 and 9), and long-term memory (Figure 10).

Question: count the number of patients who were prescribed aspirin ec within 2 month after having received a venous cath nec procedure until 4 year ago.

EHRAgent

Knowledge:
- As aspirin ec is a drug, we can find the corresponding information in the prescriptions database.
- As venous cath nec is a procedure, we can find the corresponding ICD9_CODE in the d_icd_procedures database.
- The ICD9_CODE can be used to find the corresponding HADM_ID in the procedures_icd database.
- The HADM_ID can be used to find the corresponding SUBJECT_ID in the admissions database.
- We can then count the number of unique SUBJECT_IDs in the prescriptions database who were prescribed aspirin ec within 2 months after having received a venous cath nec procedure until 4 years ago.

Figure 7: Case study of medical knowledge injection in `EHRAgent` on MIMIC-III dataset. Given a question related to *'aspirin ec'* and *'venous cath nec'*, `EHRAgent` effectively integrates knowledge about their potential location in the database and the identifiers required to associate them.

Question: what was the amount of pioglitazone that patient 16992 were prescribed in total?

ReAct

```
...
Observation 3: The corresponding HADM_ID are 155667, 107483.
Thought 4: We need to find the related information from
prescriptions database.
Action 4: LoadDB[prescriptions]
Observation 4: We have successfully loaded the prescriptions
database, including the following columns: Please modify it.
ROW_ID, SUBJECT_ID, HADM_ID, STARTDATE, ENDDATE, DRUG,
DOSE_VAL_RX, DOSE_UNIT_RX, ROUTE
Thought 5: Then, we need to filter information related to
HADM_ID=155667 and DRUG=pioglitazone.
Action 5: FilterDB[HADM_ID=155667, DRUG=pioglitazone]
Observation 5: We have successfully filtered the data (1 rows).
Thought 6: Then we need to get the DOSE_VAL_RX value.
Action 6: FilterDB[DOSE_VAL_RX, sum]
Observation 6: The corresponding DOSE_VAL_RX value is 6.0.
Thought 7: We need to find the related information from
prescriptions database.
Action 7: LoadDB[prescriptions]
Observation 7: We have successfully loaded the prescriptions
database, including the following columns: Please modify it.
ROW_ID, SUBJECT_ID, HADM_ID, STARTDATE, ENDDATE, DRUG,
DOSE_VAL_RX, DOSE_UNIT_RX, ROUTE
Thought 8: Then, we need to filter information related to
HADM_ID=155667 and DRUG=pioglitazone.
Action 8: FilterDB[HADM_ID=107483, DRUG=pioglitazone]
Observation 8: We have successfully filtered the data (1 rows).
Thought 9: Then we need to get the DOSE_VAL_RX value.
Action 9: FilterDB[DOSE_VAL_RX, sum]
Observation 9: The corresponding DOSE_VAL_RX value is 9.0.
...
```

EHRAgent

```
# We can find the visiting information of patient 16992 in the
admissions database.
patient_db = LoadDB('admissions')
filtered_patient_db = FilterDB(patient_db, 'SUBJECT_ID=16992')
print(len(filtered_patient_db))
hadm_id_list = GetValue(filtered_patient_db, 'HADM_ID, list')
print(hadm_id_list)

# As pioglitazone is a drug, we can find the corresponding
information in the prescriptions database.
prescriptions_db = LoadDB('prescriptions')

# We can find the total amount of dose of pioglitazone that
patient 16992 were prescribed by summing up the DOSE_VAL_RX in
the prescriptions database.
total_dose = 0
for hadm_id in hadm_id_list:
    filtered_prescriptions_db = FilterDB(prescriptions_db,
'HADM_ID={}||DRUG=pioglitazone'.format(hadm_id))
    dose_list = GetValue(filtered_prescriptions_db, 'DOSE_VAL_RX,
list')
    print(dose_list)
    for dose in dose_list:
        total_dose += float(dose)

answer = total_dose
```

Figure 8: Case study 1 of code interface in EHRAgent on MIMIC-III Dataset. The baseline approach, ReAct, lacks a code interface and encounters limitations when performing identical operations on multiple sets of data. It resorts to generating repetitive action steps iteratively, leading to an extended solution trajectory that may exceed the context limitations. In contrast, EHRAgent leverages the advantages of code structures, such as the use of 'for loops', to address these challenges more efficiently and effectively. The steps marked in red on the left side indicate the repeated actions by ReAct, while the steps marked in green are the corresponding code snippets by EHRAgent. By comparing the length and number of steps, the code interface can help EHRAgent save much context space.

Question: Show me patient 28020's length of stay of the last hospital stay.

ReAct

```
Thought 1: First, we need to find the last hospital stay of
patient 28020.
Action 1: LoadDB[admissions]
Observation 1: We have successfully loaded the admissions
database, including the following columns: ROW_ID, SUBJECT_ID,
HADM_ID, ADMITTIME, DISCHTIME, ADMISSION_TYPE,
ADMISSION_LOCATION, DISCHARGE_LOCATION, INSURANCE, LANGUAGE,
MARITAL_STATUS, ETHNICITY, AGE.
Thought 2: Next, we need to filter the data to only include
records for patient 28020.
Action 2: FilterDB[SUBJECT_ID=28020, max(DISCHTIME)]
Observation 2: We have successfully filtered the data (1 rows).
Thought 3: Now we have the last hospital stay record for patient
28020. We need to find the length of stay for this stay.
Action 3: GetValue[DISCHTIME-ADMITTIME, duration]
Observation 3: The value you are querying does not exist. Please
modify it.
Thought 4: It seems there was an error in step 3. Let's correct
the mistake and try again.
Action 4: GetValue[DISCHTIME - ADMITTIME, Calendar]
Observation 4: The value you are querying does not exist. Please
modify it.
Thought 5: It seems that the calculation of the length of stay
is not possible using the given steps. I apologize for the
inconvenience.
Action 5: Finish[None]
Observation 5: Answer is INCORRECT
```

EHRAgent

```
# Import the datetime module
import datetime

# Load the admissions
databaseadmissions_db = LoadDB('admissions')

# Filter the database for patient 28020
filtered_admissions_db = FilterDB(admissions_db,
'SUBJECT_ID=28020')

# Get the ADMITTIME and DISCHTIME of the last hospital
stayadmit_disch_tuple = SQLInterpreter('SELECT ADMITTIME,
DISCHTIME FROM admissions WHERE SUBJECT_ID=28020 ORDER BY
ADMITTIME DESC LIMIT 1')

# Extract the ADMITTIME and DISCHTIME from the tuple
admit_time, disch_time = admit_disch_tuple[0]

# Convert the ADMITTIME and DISCHTIME to datetime
objectsadmit_time = datetime.datetime.strptime(admit_time, '%Y-
%m-%d %H:%M:%S')
disch_time = datetime.datetime.strptime(disch_time, '%Y-%m-%d
%H:%M:%S')

# Calculate the length of hospital stay
length_of_stay = disch_time - admit_time

# Store the answer
answer = length_of_stay.days
```

Figure 9: Case study 2 of code interface in EHRAgent on MIMIC-III Dataset. When encountering challenges in tool use, ReAct will keep making trials and can be stuck in the modification process. On the other hand, with code interface, EHRAgent can take advantage of Python built-in functions to help with debugging and code modification.

Question: count the number of times that patient 85895 received a ph lab test last month.

Original Examples

```
Question: What is the maximum total hospital cost that involves
a diagnosis named comp-oth vasc dev/graft since 1 year ago?
Knowledge: {KNOWLEDGE}
Solution: {SOLUTION}

Question: Had any tpn w/lipids been given to patient 2238 in
their last hospital visit?
Knowledge: {KNOWLEDGE}
Solution: {SOLUTION}

Question: What was the name of the procedure that was given two
or more times to patient 58730?
Knowledge: {KNOWLEDGE}
Solution: {SOLUTION}

Question: What was the last time patient 4718 had a peripheral
blood lymphocytes microbiology test in the last hospital visit?
Knowledge: {KNOWLEDGE}
Solution: {SOLUTION}
```

Examples from Long-Term Memory

```
Question: Count the number of times that patient 52898 were
prescribed ns this month.
Knowledge: {KNOWLEDGE}
Solution: {SOLUTION}

Question: Count the number of times that patient 14035 had a d10w
intake.
Knowledge: {KNOWLEDGE}
Solution: {SOLUTION}

Question: Count the number of times that patient 99791 received a
op red-int fix rad/ulna procedure.
Knowledge: {KNOWLEDGE}
Solution: {SOLUTION}

Question: Count the number of times that patient 54825 received a
rt/left heart card cath procedure last year.
Knowledge: {KNOWLEDGE}
Solution: {SOLUTION}
```

Figure 10: Due to the constraints of limited context length, we are able to provide only a limited number of examples to guide EHRAgent in generating solution code. For a given question, the initial set of examples is pre-defined and fixed, which may not cover the specific reasoning logic or knowledge required to solve it. From the original examples on the left, none of the questions related to either 'count the number' scenarios or procedure knowledge. In contrast, when we retrieve examples from the long-term memory, the new set is exclusively related to 'count the number' questions, thus providing a similar solution logic for reference.