EHR-EVO: EVOLVING HARNESSES FOR EHR AGENTS THROUGH EXECUTION EXPERIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents provide a promising interface for accessing electronic health records (EHRs), but reliable execution remains difficult because database queries require complex schema navigation, tool use, and multi-step program construction. Existing systems can recover from individual failures through retries and execution feedback, yet typically rely on fixed execution scaffolds and make limited use of accumulated experience. We introduce `EHR-Evo`, an experience-driven execution harness around a frozen language-model agent. An outer-loop proposer uses sanitized execution traces to evolve model-external planning, retrieval, memory, and recovery mechanisms, after which the selected harness configuration is frozen for evaluation. During execution, the harness retrieves reusable operational knowledge and solution traces, constructs question-specific guidance, and uses execution feedback for targeted recovery and cross-question experience reuse. We evaluate `EHR-Evo` on MIMIC-III, eICU, and TREQS, comparing the complete evolved harness with fixed-retry, persistent-memory, and existing agent/harness baselines. The results show consistent improvements in answer success while revealing distinct effects of recovery and persistent experience on execution completion. Our study demonstrates that EHR-agent capability can be improved by evolving and adapting the model-external harness while keeping the underlying solver frozen.

1 INTRODUCTION

Electronic health records (EHRs) contain the information needed to answer questions about individual patients and patient populations in clinical practice and research (Johnson et al., 2016; Pollard et al., 2018; Wang et al., 2020). Accessing this information, however, often requires specialized database knowledge, programming expertise, or support from data engineers (Shi et al., 2024). LLM-based natural-language interfaces (Achiam et al., 2023; Team et al., 2023; Liu et al., 2024) offer a promising way to reduce this technical barrier by translating users' information needs into executable database operations. Reliable execution remains challenging because longitudinal patient information is distributed across admissions, diagnoses, medications, laboratory measurements, and other events. An agent must identify the relevant tables, resolve identifiers, connect records across encounters, and correctly apply temporal, filtering, and aggregation constraints (Lee et al., 2022). As a result, failures may arise from both high-level reasoning errors and low-level mistakes in schema usage, tool invocation, or query construction.

An agent that repeatedly operates in the same EHR environment has access to another source of capability beyond its model parameters: accumulated execution experience. Previous attempts reveal schema relations, tool-use conventions, effective query patterns, and recurring failure modes. Execution feedback can help recover from an unsuccessful attempt within the current question, while accumulated experience can also expose weaknesses in the surrounding execution harness and provide reusable guidance for later questions. These forms of adaptation are related but distinct: retrying can improve the current attempt, but does not determine how execution guidance should be organized, what experience should be retained, or when it should be reused. Existing approaches such as ReAct (Yao et al., 2022), self-debugging (Chen et al., 2024), EHRAgent (Shi et al., 2024), and automatic harness optimization methods (Lee et al., 2026; Lou et al., 2026; Chen et al., 2026) provide useful mechanisms for interactive execution, recovery, and harness optimization. Our focus is a complementary question: can execution experience improve a frozen agent at two timescales—by

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

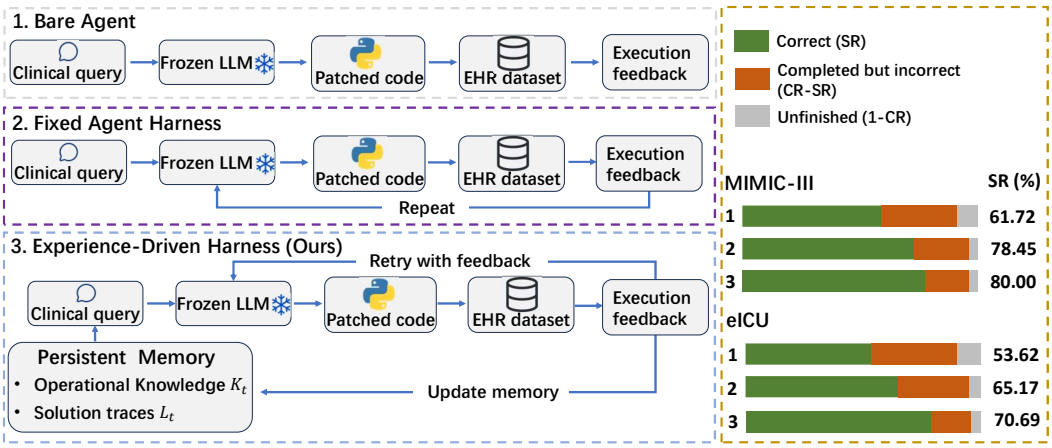

Figure 1: Comparison of EHR agent harnesses with a frozen LLM: the bare agent executes without recovery, the fixed harness introduces feedback-guided retries, and our evolved harness combines improved execution guidance, recovery, and cross-query memory. Right: success rates on MIMIC-III and eICU improve across these designs. Green, orange, and gray indicate correct, completed-but-incorrect, and unfinished cases, respectively.

shaping its model-external harness before evaluation and by providing reusable operational experience across questions after the harness is selected? Figure 1 summarizes this distinction. A bare agent executes each question without recovery, while a fixed harness introduces feedback-guided retries. Our experience-driven harness additionally evolves its execution scaffold from search-stage traces and supports reusable operational experience across questions, yielding further improvements on both MIMIC-III and eICU.

We introduce `EHR-Evo`, a memory-guided, self-evolving execution harness around a frozen database agent. Before sealed evaluation, an outer-loop proposer analyzes sanitized search-stage traces and generates candidate harness configurations over planning, retrieval, memory, and recovery behavior. Candidate configurations are evaluated externally, and the selected harness $\phi^\star$ is then frozen. Under this harness, each question is solved using schema information and retrieved experience to construct a memory plan describing relevant tables, execution steps, query patterns, and recovery hints. The agent then generates and executes database programs, using the previous program and available tool feedback when a retry is allocated. Across questions, persistent memory maintains reusable operational knowledge and selected solution traces that can guide subsequent planning and recovery. In our setting, *self-evolving* therefore refers to both outer-loop harness evolution and experience accumulation under the selected harness; the underlying EHR solver remains frozen throughout.

We evaluate `EHR-Evo` on MIMIC-III, eICU, and TREQS through comparisons among baseline execution, feedback-guided recovery, persistent-memory variants, and the complete evolved harness. These comparisons allow us to examine the different roles of within-question recovery and cross-question experience reuse. We report answer success and program completion separately to distinguish execution failures from completed but incorrect outputs.

Our contributions are threefold.

1. *Experience-driven EHR harness evolution.* We introduce an outer-loop procedure that uses search-stage execution experience to evolve model-external planning, retrieval, memory, and recovery mechanisms around a frozen EHR agent, with the selected harness frozen before sealed evaluation.
2. *Cross-question experience accumulation.* Under the selected harness, we organize reusable operational knowledge alongside selected solution traces and retrieve this experience for planning and feedback-guided recovery, allowing execution experience acquired from earlier questions to guide subsequent questions without updating the underlying solver.
3. *Evaluation of evolved EHR harnesses.* We evaluate `EHR-Evo` and its component configurations on MIMIC-III, eICU, and TREQS, measuring both answer success and execution completion and showing consistent gains over reproduced EHR-agent and harness-level baselines. The anonymous code of our framework is accessible at `https://anonymous.4open.science/r/EvoH-14B4`.

2 Related Work

2.1 EHR question answering and agent memory.

EHR question answering has been studied through text-to-SQL benchmarks such as EHRSQL, which require schema linking, temporal reasoning, and multi-table operations (Lee et al., 2022). Beyond direct query generation, tool-using agents address multi-step tasks through reasoning–action loops and execution-feedback program repair (Yao et al., 2022; Chen et al., 2024). EHRAgent applies this paradigm to EHR databases by combining domain metadata, interactive coding, error tracing, and retrieval of successful demonstrations (Shi et al., 2024). These approaches establish execution feedback and prior examples as useful sources of guidance for database reasoning. More broadly, agent-memory systems study how interaction history can support long-term behavior through episodic retrieval, reflection, and persistent context (Park et al., 2023; Zhong et al., 2024; Shinn et al., 2023). Recent work in medical agents further explores extracting reusable skills from previous experience to improve subsequent reasoning (Sun et al., 2026). `EHR-Evo` builds on this direction but focuses specifically on *operational experience* accumulated while interacting with an EHR database. Rather than representing experience only through complete past executions, the harness maintains reusable schema relations, tool-use conventions, and query patterns alongside selected solution traces. This separation allows task-specific examples to provide concrete demonstrations while more general operational knowledge can be reused across questions with different entities and temporal constraints.

2.2 Automatic harness evolution.

Model-external adaptation has traditionally focused on optimizing prompts and demonstrations. APE and OPRO formulate instruction generation as black-box optimization (Zhou et al., 2022; Yang et al., 2024); EvoPrompt and Promptbreeder apply evolutionary search (Guo et al., 2024; Fernando et al., 2023); and ProTeGi and TextGrad use textual feedback to guide iterative revisions (Pryzant et al., 2023; Yuksekgonul et al., 2024). DSPy and MIPRO extend this perspective to the joint optimization of instructions and demonstrations within multi-stage LM programs (Khattab et al., 2023; Opsahl-Ong et al., 2024). More recent work broadens the optimization target from prompts to the agent harness itself, including tools, memory, middleware, and control logic (Lee et al., 2026; Chen et al., 2026; Lin et al., 2026; Lou et al., 2026). Together, these approaches demonstrate that capability can be improved by adapting the execution scaffolding around a model without modifying its parameters. `EHR-Evo` builds on this direction but differs in how execution experience is used across two timescales. Unlike harness optimization approaches that primarily use execution evidence to select or improve the execution scaffold, `EHR-Evo` treats experience both as evidence for pre-evaluation harness search and as persistent state for post-selection cross-question adaptation, while holding the underlying EHR solver fixed. In the outer loop, a proposer analyzes sanitized execution traces and generates candidate harness configurations over planning, retrieval, memory, and recovery behavior. Candidate configurations are evaluated on search/development data, after which the selected harness is frozen for evaluation. In the inner loop, the frozen harness supports feedback-guided recovery and can reuse accumulated operational knowledge and solution traces across questions. This separation allows execution experience to affect both the scaffold under which the agent operates and the reusable guidance available to later questions, without modifying the solver parameters.

3 Preliminaries

3.1 EHR Question Answering with a Frozen Agent

We consider sequential question answering over an EHR database and its associated tools. Let $\mathcal{E}_\Sigma$ denote the database/tool environment with schema Σ, and let $\mathcal{D} = \big((q_t, a_t^\star)\big)_{t=1}^{N}$ denote an ordered sequence of questions and reference answers. The ordering is relevant because experience acquired from earlier questions may be available to later ones.

For each question q_t, a pretrained language-model agent π_θ generates and executes database programs in $\mathcal{E}_\Sigma$. A question may involve multiple attempts: $j = 0$ denotes the initial attempt and $j > 0$ denotes subsequent retries. Let $p_t^{(j)}$ denote the program generated at attempt j, and let $\tau_t^{(j)}$ denote its

execution trace, including tool interactions, outputs, and execution feedback. We use τ_t for the complete execution record of question q_t and $\hat{a}_t$ for its final answer.

The agent parameters θ remain *frozen* throughout harness search, question answering, and retry. Adaptation therefore occurs only through the model-external execution harness and its state.

We distinguish successful answering from successful execution. Let $C_t \in \{0, 1\}$ indicate whether the final program completes under the execution criterion, and define answer success as

$$J_t = C_t \ \mathrm{Judge}(\hat{a}_t, a_t^\star), \tag{1}$$

where Judge denotes the benchmark-specific answer evaluation procedure. The success rate and completion rate are

$$\mathrm{SR} = \frac{100}{N} \sum_{t=1}^{N} J_t, \qquad \mathrm{CR} = \frac{100}{N} \sum_{t=1}^{N} C_t. \tag{2}$$

Reporting both metrics distinguishes executions that fail to complete from those that complete but produce an incorrect answer.

3.2 Execution Harness and Persistent Experience

We use the term *execution harness* to denote the model-external scaffold that governs how the frozen agent receives information, retrieves prior experience, constructs execution guidance, executes programs, responds to feedback, and retains experience. Let $\mathcal{H}_\phi$ denote a harness parameterized by configuration ϕ, and let M_t denote its persistent state before processing question q_t.

We distinguish two time scales of harness adaptation. At the *outer* time scale, candidate harness configurations are proposed and evaluated before sealed evaluation:

$$\phi^{(0)}, \phi^{(1)}, \ldots, \phi^{(K)} \ \longrightarrow \ \phi^\star, \tag{3}$$

where $\phi^\star$ denotes the selected harness configuration. Candidate configurations may modify model-external components such as planning guidance, retrieval policies, memory organization, and runtime-recovery behavior. The selected configuration $\phi^\star$ is frozen before sealed evaluation.

At the *inner* time scale, the frozen harness $\mathcal{H}_{\phi^\star}$ processes a sequence of EHR questions. At attempt j, it constructs an execution context from the current question, database schema, persistent state, and traces from previous attempts:

$$c_t^{(j)} = \mathcal{H}_{\phi^\star}\big(q_t, \Sigma, M_t, \tau_t^{(<j)}\big), \tag{4}$$

$$p_t^{(j)} \sim \pi_\theta\big(\cdot \mid c_t^{(j)}\big), \tag{5}$$

where $\tau_t^{(<j)}$ is empty for the initial attempt. Executing $p_t^{(j)}$ in $\mathcal{E}_\Sigma$ produces the corresponding trace $\tau_t^{(j)}$.

This formulation separates two forms of adaptation during execution. Within a question, previous traces $\tau_t^{(<j)}$ provide execution feedback that can guide recovery. Across questions, the persistent state M_t can retain information acquired from earlier interactions and make it available to subsequent questions. The former enables *within-question recovery*, whereas the latter enables *cross-question experience reuse*.

More generally, persistent state evolves under the selected harness as

$$M_{t+1} = \mathrm{Update}_{\phi^\star}\big(M_t, q_t, \tau_t, f_t\big), \tag{6}$$

where f_t denotes the feedback available under the execution protocol. When persistent updates are disabled, $M_{t+1} = M_t$.

The resulting framework therefore contains two distinct adaptation mechanisms. Outer-loop harness evolution searches over the configuration ϕ before evaluation, while inner-loop experience accumulation updates the persistent state M_t across questions under the selected and frozen configuration $\phi^\star$. The EHR solver parameters θ remain fixed in both stages. During sealed evaluation, neither the harness proposer nor the harness configuration is updated; only the persistent state changes when the evaluated configuration permits memory write-back. In the following section, we introduce `EHR-Evo` as a concrete instantiation of this two-timescale formulation.

4 METHODOLOGY

Figure 2 illustrates EHR-Evo, an experience-driven execution harness around a frozen EHR database agent. The framework operates at two levels. Before evaluation, an outer-loop proposer uses sanitized execution traces to generate candidate harness configurations, which are evaluated on designated search/development data. The selected configuration $\phi^\star$ is then frozen. During evaluation, the harness retrieves applicable operational knowledge and previous solution traces to construct question-specific execution guidance, uses execution feedback for recovery, and consolidates selected experience into persistent memory for subsequent questions. Throughout this process, the EHR solver remains frozen.

4.1 OUTER-LOOP HARNESS EVOLUTION

Starting from an initial harness configuration $\phi^{(0)}$, the outer-loop proposer generates candidate configurations using sanitized execution traces:

$$\tilde{\phi}^{(k+1)} = \mathrm{Propose}\big(\phi^{(k)}, \bar{\mathcal{T}}^{(k)}\big), \tag{7}$$

where $\bar{\mathcal{T}}^{(k)}$ denotes the available search-stage execution evidence.

Candidate configurations may modify model-external components such as planning guidance, retrieval policies, memory organization, and runtime-recovery settings. They are evaluated on designated search/development data, and the selected configuration is

$$\phi^\star = \mathrm{Select}\big(\phi^{(0)}, \tilde{\phi}^{(1)}, \ldots, \tilde{\phi}^{(K)}\big). \tag{8}$$

The resulting $\phi^\star$ is frozen before sealed evaluation; the outer proposer is not invoked while answering evaluation questions.

4.2 EHR-GROUNDED OPERATIONAL MEMORY

We follow the notation introduced in Section 3. Under the selected harness $\mathcal{H}_{\phi^\star}$, before processing question q_t, the harness maintains a persistent memory

$$M_t = (K_t, L_t), \tag{9}$$

where K_t stores reusable operational knowledge and L_t stores selected solution traces from previous questions.

The two memory components serve complementary roles. The operational store K_t captures database-specific knowledge that can remain useful across questions, including schema relations, tool-use conventions, identifier relationships, and reusable query patterns. Records may additionally contain information about their source, applicability, and available supporting or contradictory evidence.

The solution store L_t, in contrast, retains concrete execution traces. Such traces illustrate how database operations were combined to solve particular questions, but may also contain details specific to a patient, encounter, or temporal condition. Separating the two forms of memory allows reusable operational knowledge to transfer across questions without treating every previous solution as directly applicable.

During execution, the harness also maintains question-local context describing the current patient state, task progress, and recovery status. These states support the current execution but are not additional cross-question stores. Persistent experience accumulation is represented by updates to K_t and L_t.

4.3 MEMORY-GUIDED EXECUTION AND RECOVERY

Adaptive memory retrieval. For each question q_t, the harness retrieves experience applicable to the current database operation:

$$R_t = \mathrm{Retrieve}_{\phi^\star}\big(q_t, \Sigma, M_t\big), \tag{10}$$

where R_t may contain records from both K_t and L_t. Retrieval combines lexical relevance with structural compatibility, including query family, relevant tables, encounter scope, and tool-use information.

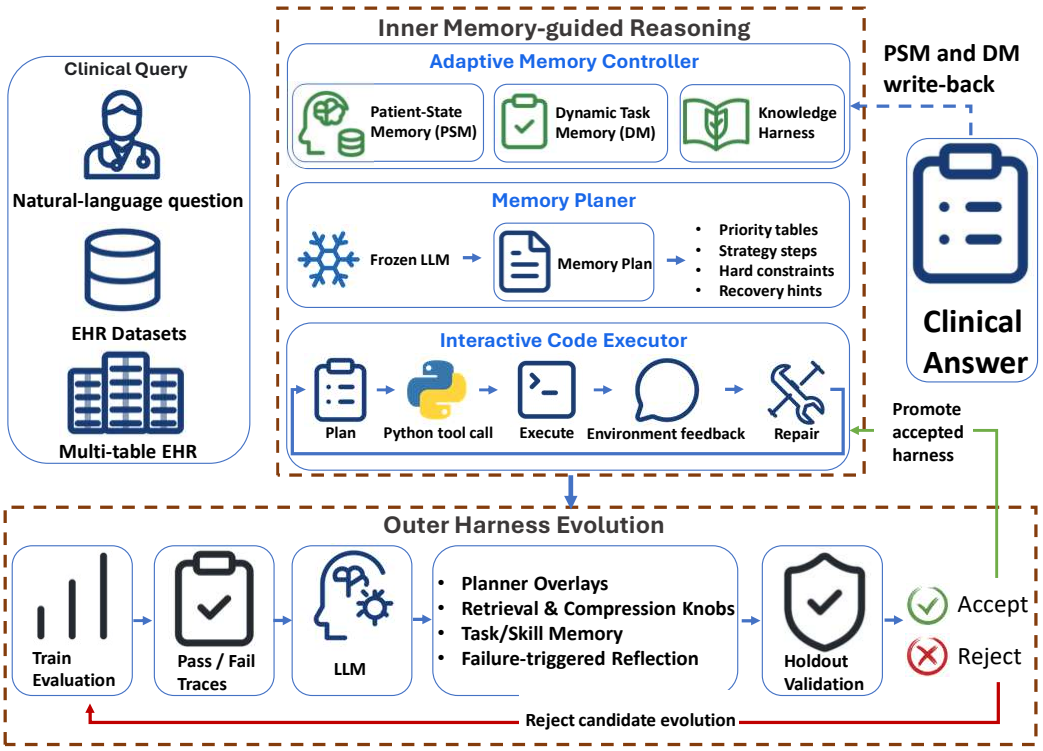

Figure 2: **Outer-loop harness evolution and memory-guided execution in `EHR-Evo`.** The outer loop proposes and evaluates candidate harness configurations using sanitized search-stage traces, after which the selected harness $\phi^\star$ is frozen. During evaluation, the frozen EHR solver retrieves operational knowledge and solution traces, executes database programs, uses feedback for runtime recovery, and writes selected experience back to persistent memory for subsequent questions.

Memory-guided planning. The retrieved experience is organized into a compact execution plan:

$$\mathcal{P}_t = \mathrm{Plan}_{\phi^\star}\big(q_t, \Sigma, R_t\big). \tag{11}$$

The plan summarizes relevant tables, likely execution steps, question-specific constraints, and available recovery hints. It exposes accumulated experience to the frozen agent without fixing the final program in advance.

Interactive execution. At attempt j, the harness assembles the execution context as

$$c_t^{(j)} = \mathrm{Assemble}_{\phi^\star}\big(q_t, \Sigma, \mathcal{P}_t, R_t, \tau_t^{(<j)}, \rho_t^{(j)}\big), \tag{12}$$

$$p_t^{(j)} \sim \pi_\theta\big(\cdot \mid c_t^{(j)}\big), \tag{13}$$

where $\rho_t^{(0)} = \varnothing$. Executing $p_t^{(j)}$ produces the corresponding trace $\tau_t^{(j)}$, including tool interactions, outputs, and execution feedback.

A top-level retry is distinct from the ordinary internal agent–tool interaction loop: it begins a new attempt using evidence produced by the preceding execution.

Feedback-guided recovery. When a retry is allocated, `EHR-Evo` constructs a targeted repair context from the preceding program, observed tool feedback, and relevant operational memory. Let $e_t^{(j)}$ denote the available execution error or diagnostic signal. The harness assigns an operational failure category

$$z_t^{(j)} = \mathrm{Classify}\big(e_t^{(j)}, p_t^{(j)}\big), \tag{14}$$

and constructs

$$\rho_t^{(j+1)} = \mathrm{Repair}_{\phi^\star}\big(q_t, p_t^{(j)}, e_t^{(j)}, z_t^{(j)}, R_t\big). \tag{15}$$

The failure categories capture recurring operational problems such as syntax errors, identifier or return-type mismatches, missing columns, invalid conditions, and date-time handling errors. The retry therefore combines local execution evidence with experience accumulated from earlier questions.

4.4 Cross-Question Experience Consolidation

After processing q_t, the harness determines which parts of the resulting experience should influence future questions. We separate *solution-trace retention* from *operational-knowledge consolidation*.

Solution-trace retention. Let $\mathcal{W}_{\phi^\star}$ denote the configured trace-admission policy:

$$W_t = \mathcal{W}_{\phi^\star}(G_t, J_t), \tag{16}$$

where G_t summarizes available execution evidence. The solution memory is updated as

$$L_{t+1} = \begin{cases} \mathrm{Consolidate}_L\big(L_t, q_t, \tau_t^\star\big), & W_t = 1, \\ L_t, & W_t = 0. \end{cases} \tag{17}$$

Operational-knowledge consolidation. Operational evidence follows a separate update path:

$$K_{t+1} = \mathrm{UpdateKnowledge}_{\phi^\star}\big(K_t, q_t, \tau_t, f_t\big), \tag{18}$$

where f_t denotes the available execution outcomes and feedback. This update may add newly observed schema or tool-use information, strengthen existing records, or retain contradiction evidence.

Together, the two update paths define

$$M_t = (K_t, L_t) \quad \longrightarrow \quad M_{t+1} = (K_{t+1}, L_{t+1}). \tag{19}$$

Thus, the harness configuration $\phi^\star$ remains fixed during sealed evaluation, while its persistent state M_t can evolve across questions. When memory updates are disabled, $M_{t+1} = M_t$.

5 Experiments

5.1 Experimental Setup

Datasets and implementation. We evaluate on three settings used by EHRAgent (Shi et al., 2024): MIMIC-III (581 questions), eICU (580), and TREQS (996). MIMIC-III and eICU contain locally preprocessed EHRSQL questions over single-center hospital and multi-center critical-care records, respectively (Johnson et al., 2016; Pollard et al., 2018; Lee et al., 2022), while TREQS is derived from MIMICSQL (Wang et al., 2020). We retain Levels I–IV for MIMIC-III and Levels I–III for eICU and TREQS.

Using the EHRAgent environment, the frozen EHR solver generates Python programs that invoke dataset-specific database tools and store results in `answer`. Model parameters are never updated; adaptation occurs through the external execution harness and its persistent state. Further details appear in Appendices B.1, B.2, and B.3.

Model roles and harness search. GPT-5.6-sol serves as the outer-loop harness proposer, using sanitized search-stage traces to generate candidate configurations over planning, retrieval, memory, scope constraints, and runtime recovery. Candidates are evaluated on search/development data, and the selected configuration $\phi^\star$ is frozen before sealed evaluation.

GPT-4-0613 (Achiam et al., 2023) serves as the frozen EHR solver for question answering, program generation, tool interaction, and runtime retry. GPT-5.6-sol is not invoked during sealed evaluation and does not observe evaluation-time failures. Thus, harness-configuration search occurs before evaluation, while all final per-question reasoning and repair are performed by the same frozen GPT-4-0613 solver under the selected harness.

Configurations and protocol. We compare the EHRAgent baseline with three `EHR-Evo` configurations. *`EHR-Evo` (w/o memory)* retains feedback-guided recovery but disables cross-question memory growth, while *`EHR-Evo` (w/o retry)* retains persistent operational knowledge and solution-memory updates without the additional top-level retry. The full `EHR-Evo` combines both mechanisms.

The component variants characterize immediate recovery and persistent experience reuse. For the controlled TREQS comparison, the retry-only and memory/knowledge runs start from the same four seed demonstrations and process all 996 TREQS questions in the same fixed order. The two runs

maintain separate persistent states, so experience accumulated in one configuration is not shared with the other.

Benchmark judgments control retry allocation and, for Memory/Knowledge, solution-memory admission, although reference-answer text is not inserted into repair prompts. The resulting protocol is therefore benchmark-conditioned rather than fully reference-free. Further details are given in Appendices B.3 and B.4.

Baselines. We include published EHRAgent results for CoT (Wei et al., 2022), Self-Consistency (Wang et al., 2022), Chameleon (Lu et al., 2023), ReAct (Yao et al., 2022), Reflexion (Shinn et al., 2023), LLM2SQL (Nan et al., 2024), DIN-SQL (Pourreza & Rafiei, 2023), Self-Debugging (Chen et al., 2024), AutoGen (Wu et al., 2023), and EHRAgent (Shi et al., 2024). We additionally include Meta-Harness (Lee et al., 2026) as a harness-level baseline.

Our EHRAgent rerun obtains overall SRs of 61.72%, 53.62%, and 66.16% on MIMIC-III, eICU, and TREQS, compared with 58.97%, 53.10%, and 69.70% reported in the original work. This close agreement provides a reproduction sanity check, and we use the rerun as the primary internal baseline. For harness-level comparison, GPT-5.6-sol is used as the outer proposer while GPT-4-0613 remains the frozen EHR solver, separating proposer capability from the execution model being evaluated.

Metrics. We report success rate (SR) and completion rate (CR):

$$\mathrm{SR} = \frac{100}{N}\sum_{t=1}^{N} J_t, \qquad \mathrm{CR} = \frac{100}{N}\sum_{t=1}^{N} C_t, \tag{20}$$

where J_t denotes benchmark answer success and C_t denotes execution completion. Reporting both separates unfinished executions from completed but incorrect answers. Neither metric establishes clinical correctness; evaluation details appear in Appendix B.5.

5.2 Comparison with Existing Methods

Table 1 compares published baselines with our reruns across MIMIC-III, eICU, and TREQS. Published GPT-4-0613 results provide literature context, while our EHRAgent rerun serves as the primary internal baseline. The harness-level methods are evaluated with the same frozen EHR solver, so improvements reflect changes in the model-external execution scaffold rather than parameter updates to the solver. More experiments can be found in Appnedix C.

Overall performance. The full `EHR-Evo` achieves overall SRs of 80.00%, 70.69%, and 79.62% on MIMIC-III, eICU, and TREQS, respectively. Relative to the EHRAgent rerun, these correspond to gains of 18.28, 17.07, and 13.46 percentage points; the margins over Meta-Harness are 22.07, 28.28, and 17.57 points. The gains are consistent across datasets with different schemas and query distributions. These results show that a frozen EHR solver can benefit substantially from model-external harness evolution together with reusable execution experience, without updating the solver itself.

Answer success versus completion. Higher completion does not necessarily imply higher answer success. On TREQS, Meta-Harness reaches 95.98% CR but 62.05% SR, compared with 90.08% CR and 79.62% SR for `EHR-Evo`. Similarly, on eICU, `EHR-Evo` (w/o retry) has higher CR than the full configuration (95.69% vs. 93.68%) but lower SR (63.79% vs. 70.69%). This difference indicates that reliably terminating an execution is not sufficient for producing a benchmark-correct answer and motivates reporting both metrics.

Performance across query complexity. `EHR-Evo` achieves 95.29/85.41/80.50/64.24% SR across MIMIC-III Levels I–IV, 75.00/76.69/33.33% on eICU Levels I–III, and 80.95/79.06/59.09% on TREQS Levels I–III. Performance decreases on the hardest queries, indicating that harness adaptation improves execution robustness without eliminating the difficulty of multi-step database reasoning. TREQS Level III contains only 22 questions, so its result should be interpreted together with the corresponding sample size.

5.3 Retry and Cross-Question Experience Reuse

The component configurations provide complementary evidence about how execution experience is used over different time scales. The w/o-memory configuration retains feedback-guided recovery

Table 1: Results on MIMIC-III, eICU, and TREQS (%). Per-level columns report SR; overall columns report SR and CR. † denotes published results from Shi et al. (2024) using GPT-4-0613. The lower block reports our reruns. Our EHRAgent reproduction closely matches the published overall performance and is used as the primary internal baseline for comparison with the evaluated harness configurations. Component variants characterize the behavior of retry and persistent experience reuse.

Dataset	MIMIC-III						eICU					TREQS				
Method	**I**	**II**	**III**	**IV**	**SR**	**CR**	**I**	**II**	**III**	**SR**	**CR**	**I**	**II**	**III**	**SR**	**CR**
Methods without a code interface																
CoT†	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†	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†	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†	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†	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
Methods with a code interface																
LLM2SQL†	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
DIN-SQL†	49.51	44.22	36.25	21.85	38.45	81.72	23.49	26.13	12.50	25.00	55.00	41.34	36.38	12.73	38.05	82.73
Self-Debugging†	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†	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†	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
Our reruns																
EHRAgent	76.19	66.13	58.49	51.66	61.72	89.46	63.97	56.10	22.67	53.62	83.62	71.54	61.59	18.75	66.16	89.46
Meta-Harness	75.00	67.20	55.97	39.07	57.93	96.72	54.41	43.36	16.00	42.41	88.45	60.76	64.81	36.36	62.05	**95.98**
EHR-Evo (w/o memory)	90.59	83.78	78.62	**64.90**	78.45	**97.59**	69.85	70.46	30.67	65.17	92.76	**84.63**	73.07	18.75	78.31	89.86
EHR-Evo (w/o retry)	90.48	83.87	**81.76**	63.58	78.97	96.55	67.64	69.65	28.00	63.79	**95.69**	80.19	73.72	45.45	76.51	87.24
EHR-Evo	**95.29**	**85.41**	80.50	64.24	**80.00**	97.41	**75.00**	**76.69**	**33.33**	**70.69**	93.68	80.95	**79.06**	**59.09**	**79.62**	90.08

within the current question, whereas the w/o-retry configuration relies more directly on experience accumulated from previous questions. Across the three datasets, both variants substantially outperform the EHRAgent rerun. The w/o-memory configuration improves overall SR from 61.72% to 78.45% on MIMIC-III, 53.62% to 65.17% on eICU, and 66.16% to 78.31% on TREQS. The w/o-retry configuration reaches 78.97%, 63.79%, and 76.51%, respectively. These results indicate that both immediate execution recovery and persistent experience reuse can provide substantial gains around the same frozen agent.

Their relative behavior differs across datasets. The w/o-retry configuration is slightly higher on MIMIC-III (78.97% versus 78.45%), whereas w/o-memory performs better on eICU (65.17% versus 63.79%) and TREQS (78.31% versus 76.51%). At TREQS Level III, w/o-retry achieves 45.45% compared with 18.75% for w/o-memory, although this level contains only 22 questions and should therefore be interpreted cautiously. The full EHR-Evo obtains the strongest overall SR among the recorded configurations on all three datasets: 80.00% on MIMIC-III, 70.69% on eICU, and 79.62% on TREQS. This pattern is consistent with the motivation for combining local recovery with reusable execution experience: feedback addresses failures in the current query, while persistent memory makes previously acquired schema, tool-use, and query knowledge available to subsequent questions.

6 Conclusion

We introduced EHR-Evo, an experience-driven framework that combines outer-loop harness evolution with memory-guided execution around a frozen EHR agent. Execution traces guide the evolution of model-external planning, retrieval, memory, and recovery mechanisms, after which the selected harness is frozen for evaluation. During execution, feedback-guided recovery and persistent operational experience support adaptation without updating the underlying solver. Experiments on MIMIC-III, eICU, and TREQS demonstrate consistent improvements while also distinguishing answer success from execution completion. Several limitations remain. First, benchmark judgments guide retry allocation and, in some settings, memory admission, providing feedback unavailable in routine deployment. Second, sequential memory accumulation depends on question order, initialization, retrieval, and write policies; error propagation and long-term stability remain insufficiently studied. Third, computational constraints restrict the model backbones, harness-search configurations, and repeated runs evaluated, limiting conclusions about generality and variance. Finally, benchmark answer matching does not establish clinical validity. Future work should prioritize reference-free feedback, broader proposer–solver combinations, repeated evaluations, and memory robustness over longer interaction sequences.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, et al. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, volume 2024, pp. 8746–8825, 2024.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 34133–34156, 2024.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):160035, 2016.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

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

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P Murphy. Autoharness: improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026.

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. *Advances in Neural Information Processing Systems*, 36:43447–43478, 2023.

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. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4556–4579, 2024.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Tom J Pollard, Alistair EW Johnson, Jesse D Raffa, Leo A Celi, Roger G Mark, and Omar Badawi. The eicu collaborative research database, a freely available multi-center database for critical care research. *Scientific data*, 5(1):180178, 2018.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in neural information processing systems*, 36:36339–36348, 2023.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7957–7968, 2023.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Haoran Sun, Wenjie Li, Yujie Zhang, Zekai Lin, Fanrui Zhang, Kaitao Chen, Xingqi He, Yichen Li, Mianxin Liu, Lei Liu, et al. Experience makes skillful: Enabling generalizable medical agent reasoning via self-evolving skill memory. *arXiv preprint arXiv:2606.09365*, 2026.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

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.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

A Execution and Memory-Update Procedure

Algorithm 1 summarizes the inner execution procedure after outer-loop harness selection. It describes memory-guided execution, feedback-guided recovery, and cross-question experience accumulation under the selected configuration $\phi^\star$. Throughout this stage, both the frozen-agent parameters θ and the selected harness configuration $\phi^\star$ remain fixed, while persistent memory may evolve across questions when the corresponding update mechanisms are enabled.

For each question q_t, the harness first retrieves relevant experience from the current persistent memory $M_t = (K_t, L_t)$ and organizes it into a memory-guided plan $\mathcal{P}_t$. The frozen agent then generates and executes a database program using this guidance. If an additional top-level attempt is allocated, execution feedback from the preceding attempt is converted into a targeted repair context and incorporated into the revised execution context.

After the final attempt is selected, the harness updates the two persistent memory components separately. The solution-memory store L_t retains selected execution traces according to the configured admission policy, whereas the operational knowledge store K_t consolidates reusable information such as schema relations, tool-use conventions, and query patterns. Consequently, an execution need not be retained as a positive solution example in order to contribute useful operational evidence. The resulting state M_{t+1} is then available when processing the next question.

The procedure exposes two temporal forms of adaptation. Within a question, M_t remains fixed while execution feedback from the current attempt supports recovery. Across questions, selected evidence may update K_t and L_t, allowing the guidance available to later questions to evolve over time. When both memory-update paths are disabled, $M_{t+1} = M_t$, reducing the procedure to fixed execution guidance with optional within-question recovery.

Importantly, Algorithm 1 describes execution under the already selected harness $\phi^\star$ and does not include the outer-loop proposer itself. The retry policy, solution-memory admission policy, and operational-knowledge update policy are controlled independently by the experimental configuration. The exact settings used in each evaluation are described in Section 5.1.

B Additional Experimental Details

B.1 Dataset Versions and Evaluation Subsets

MIMIC-III and eICU. The local preprocessed EHRSQL files contain 581 unique MIMIC-III question identifiers and 580 unique eICU identifiers. These are answerable subsets of the corresponding validation files rather than the complete original splits. Consequently, this evaluation does not measure abstention on unanswerable EHRSQL questions. A separate historical MIMIC-III runner shuffles the 581-question file with seed 42 and executes the first 580 questions; results from this protocol are distinguished from those obtained by evaluating all 581 questions.

TREQS. The TREQS backend exposes five tables: `DEMOGRAPHIC`, `DIAGNOSES`, `PROCEDURES`, `PRESCRIPTIONS`, and `LAB`. The evaluation procedure assigns complexity levels by counting distinct benchmark table names detected in the reference SQL. These levels characterize query structure rather than clinical reasoning difficulty. Levels I–III contain 525, 449, and 22 questions, respectively, giving 996 questions in total.

All reported TREQS results are obtained from fresh execution over the complete 996-question evaluation set. Each evaluated configuration independently processes all 996 questions in the same fixed order under its corresponding execution and memory settings. No outcomes from earlier baseline runs are carried forward or reused when computing the reported TREQS results.

Overall success and completion rates are computed directly from the complete set of evaluation records. Missing or unfinished executions remain in the denominator, so the reported metrics use the full 996-question evaluation set.

Algorithm 1 Memory-guided EHR execution under the selected harness

Require: Ordered questions $q_{1:N}$, environment $\mathcal{E}_\Sigma$, frozen agent π_θ, selected harness configuration $\phi^\star$, initial memory $M_1 = (K_1, L_1)$

for $t = 1$ to N **do**
  // Retrieve experience for the current question
  $R_t \leftarrow \mathrm{Retrieve}_{\phi^\star}(q_t, \Sigma, M_t)$
  $\mathcal{P}_t \leftarrow \mathrm{Plan}_{\phi^\star}(q_t, \Sigma, R_t)$
  // Initial execution
  Assemble initial context $c_t^{(0)}$ from q_t, Σ, R_t, and $\mathcal{P}_t$
  Generate and execute $p_t^{(0)} \sim \pi_\theta(\cdot \mid c_t^{(0)})$
  Record execution trace $\tau_t^{(0)}$
  // Feedback-guided recovery
  if a retry is allocated **then**
    Extract available execution feedback
    Classify the operational failure
    Construct targeted repair context $\rho_t^{(1)}$
    Assemble revised context $c_t^{(1)}$ using $\rho_t^{(1)}$ and previous execution evidence
    Generate and execute $p_t^{(1)} \sim \pi_\theta(\cdot \mid c_t^{(1)})$
    Record execution trace $\tau_t^{(1)}$
  end if
  Select final prediction and execution trace $\tau_t^\star$
  // Update retained solution experience
  if solution-memory updates are enabled **then**
    Evaluate the configured trace-admission policy
    Update L_{t+1} using Equation 17
  else
    $L_{t+1} \leftarrow L_t$
  end if
  // Consolidate reusable operational knowledge
  if operational-knowledge updates are enabled **then**
    Update K_{t+1} using Equation 18
  else
    $K_{t+1} \leftarrow K_t$
  end if
  $M_{t+1} \leftarrow (K_{t+1}, L_{t+1})$
end for

Ensure: Final predictions, execution traces, and resulting persistent memory

B.2 Runtime Configuration and Execution Budget

Shared execution settings. The preserved TREQS wrapper specifies seed 0, four requested demonstrations, and `-no_shuffle`. The run logs independently confirm a four-entry initial solution pool and the same frozen input order. The requested demonstration count does not imply that every subsequent repair prompt contains all four examples. Both configurations record heuristic planning and disabled WorldMM context.

Fixed Retry. This configuration enables `-harness_retry_on_fail`, `-memory_read_only`, and `-ltm_disable`. It permits one additional top-level repair attempt while disabling online memory growth. Existing seed retrieval and the base agent's within-attempt debugging remain available. Read-only memory therefore differs from removing memory-based retrieval altogether.

Memory/Knowledge. This configuration enables `-memory_agent` and online memory updates, without the additional top-level retry. The `-memory_trace` option records retrieval information; the option itself does not enable learning. The base agent's existing debugging remains available.

Fixed Retry and Memory/Knowledge are thus separate configurations, rather than an additive ablation that changes only memory while holding retry behavior fixed.

Scope of the retry budget. A top-level attempt can contain multiple agent and tool turns. The additional repair attempt is therefore distinct from the consecutive-auto-reply limit. In the inspected implementation, the extra-repair branch requires an available reference answer and a failed benchmark judgment. Even if the reference-answer text is absent from the repair prompt, this allocation policy remains benchmark-conditioned.

Semantic validation. Neither of the corresponding component configurations enables `-ehr_semantic_harness`. Their comparison therefore does not measure the contribution of explicit semantic validation. In particular, the reported improvements are not attributed to explicit validation of patient/admission granularity, temporal scope, or aggregation semantics.

Model configuration. The wrapper's requested model identifier is configurable. The available model-routing records for the Memory/Knowledge run refer to GPT-4-0613 service identifier. These records do not establish a common immutable model snapshot across conditions. The inspected records also do not establish all run-level coding-agent sampling parameters and token budgets. Defaults from auxiliary model calls are not substituted for these unavailable settings.

B.3 Initialization and Persistent-Memory Updates

Initialization. All evaluated configurations use the same frozen EHR solver and begin from the corresponding dataset-specific initialization. For the controlled TREQS comparison, the retry-only and Memory/Knowledge configurations start from the same four seed demonstrations and process all 996 evaluation questions in the same fixed order. The runs maintain separate persistent states, so information accumulated under one configuration is not transferred to another.

Memory configurations. The configurations reported in Table 1 differ in how persistent experience is used. `EHR-Evo` (w/o memory) retains feedback-guided recovery while disabling cross-question memory growth. `EHR-Evo` (w/o retry) enables persistent operational-knowledge and solution-memory updates but does not allocate the additional top-level retry. The full `EHR-Evo` combines feedback-guided recovery with persistent experience reuse.

Accordingly, disabled memory growth should not be interpreted as a memory-free agent: fixed seed demonstrations and any initialized read-only guidance may still be retrieved during execution. Conversely, when memory updates are enabled, selected operational evidence and solution traces may become available to subsequent questions according to the configured admission policies.

Observed performance. On MIMIC-III, `EHR-Evo` (w/o memory), `EHR-Evo` (w/o retry), and the full `EHR-Evo` achieve overall SRs of 78.45%, 78.97%, and 80.00%, respectively. On eICU, the corresponding SRs are 65.17%, 63.79%, and 70.69%, while on TREQS they are 78.31%, 76.51%, and 79.62%. The corresponding CR values are 97.59%, 96.55%, and 97.41% on MIMIC-III; 92.76%, 95.69%, and 93.68% on eICU; and 89.86%, 87.24%, and 90.08% on TREQS. These values are the configuration-level results used throughout the paper.

Memory admission. When solution-memory updates are enabled, admission follows the configured execution protocol. In the controlled TREQS Memory/Knowledge setting, benchmark judgments may be used for solution-memory admission, although reference-answer text itself is not inserted into execution or repair prompts. Operational-knowledge updates follow their configured evidence and admission rules and are treated separately from solution-trace retention.

B.4 Checkpointing and Resumption

The runner supports restoring saved solution traces and restarting from a specified question index or identifier. Evaluation records are merged by unique question identifier in the frozen input order. Selected memory stores use atomic file replacement; this does not establish transactional consistency across all memory stores and result files.

B.5 Prediction Extraction and Evaluation

Success and completion. Let $C_i \in \{0, 1\}$ indicate completion under the specified parser, and let $J_i \in \{0, 1\}$ indicate a completed answer accepted by the reference matcher. For N evaluation questions, we compute

$$\mathrm{SR} = \frac{100}{N} \sum_{i=1}^{N} J_i, \qquad \mathrm{CR} = \frac{100}{N} \sum_{i=1}^{N} C_i. \tag{21}$$

Overall metrics are question-weighted rather than unweighted averages of per-level percentages. For TREQS, the reported overall metrics are computed directly over all 996 freshly executed evaluation questions.

Answer matching. The inspected matcher uses reference-string containment with limited Boolean and numeric normalization, rather than strict denotation equality. It does not enforce element multiplicity, ordering, or the absence of extraneous answers. Reported SR must therefore be interpreted under this benchmark matcher, not as a direct measure of clinical validity or unrestricted semantic correctness.

Transcript selection. The audit removes explicit reference-answer annotations and selects the final retry transcript when an additional attempt is present. It does not select the best answer across attempts. Retaining whichever attempt matches the reference would implement a different evaluation policy.

Evaluator-version differences. Completion depends on the evaluator version. The current runner can accept selected short final Python returns without an explicit `TERMINATE` marker, whereas the earlier audit requires a standalone termination marker. Some historical scripts may also exclude missing logs from their denominators. Statistics produced by these different procedures are not silently pooled. In particular, CR is not interpreted as proof of error-free execution or a semantically correct answer.

B.6 Result Provenance and Reporting Scope

Published comparisons. Rows marked † reproduce results reported by EHRAgent (Shi et al., 2024) using GPT-4-0613. These values provide literature context and are distinct from our reruns under the present evaluation environment. Our EHRAgent reproduction closely matches the published overall performance and is therefore used as the primary internal baseline for comparison with the evaluated harness configurations.

Recorded reruns. The lower block of Table 1 reports configuration-specific reruns of EHRAgent, Meta-Harness, and `EHR-Evo`. Each result is associated with its corresponding run directory, configuration, and evaluation records. Results from different configurations are not combined merely because they appear in a shared launcher log.

For `EHR-Evo`, GPT-5.6-sol is used only in the outer harness-search stage to propose candidate configurations from sanitized search traces. The selected harness $\phi^\star$ is frozen before sealed evaluation, while GPT-4-0613 remains the frozen EHR solver for question answering, tool use, and runtime repair. Persistent-memory behavior during evaluation follows the configuration being evaluated.

Audited TREQS evaluation. The raw-log audit supports the reported fresh TREQS evaluation: 425/525 questions are successful at Level I, 355/449 at Level II, and 13/22 at Level III. These correspond to 80.95%, 79.06%, and 59.09%, respectively, giving an overall result of 793/996, or 79.62%. All 996 questions are executed under the reported configuration; no outcomes are carried forward from a previous baseline run.

Component configurations. The w/o-memory and w/o-retry variants are included to characterize two distinct adaptation mechanisms: within-question recovery and cross-question persistent experience reuse. Because the configurations differ in their enabled adaptation mechanisms, we interpret their differences descriptively rather than as isolated causal estimates of retry or memory.

Table 2: Comparison between the initial and evolved harness configurations (%). Levels I–III report SR; the final two columns report overall SR and CR.

Configuration	I	II	III	SR	CR
Initial $\phi^{(0)}$	**46.32**	41.46	9.33	38.45	88.79
Evolved $\phi^{\star}$	44.85	**45.26**	**10.67**	**40.69**	**91.03**

For the controlled TREQS comparison, the corresponding retry and memory runs use the same initial seed examples and question order, while maintaining separate persistent states. Benchmark judgments may be used for retry allocation and solution-memory admission according to the evaluated configuration; reference-answer text itself is not inserted into execution or repair prompts.

Excluded artifacts and evaluator consistency. Artifacts with incompatible tool interfaces, explicit reference-answer contamination in prediction extraction, or unresolved execution records are excluded from the reported comparisons. Metrics are associated with their corresponding evaluator and execution records; outputs from incompatible evaluator versions are not silently pooled. Schema-misaligned priors are documented separately and are not, by themselves, treated as evidence of answer leakage.

Reporting scope. The reported results support the evaluation of the complete `EHR-Evo` harness together with component variants and harness-level baselines.

The observed improvements concern database-agent execution and benchmark answer production. They should not be interpreted as evidence of newly acquired diagnostic or therapeutic knowledge, nor as validation of clinical decision-making competence.

C Effect of Outer-Loop Harness Evolution

To examine the benefit of pre-evaluation harness evolution, we compare the initial configuration $\phi^{(0)}$ with the selected configuration $\phi^{\star}$. Table 2 reports answer success across query complexity levels, together with overall success rate (SR) and completion rate (CR).

The evolved harness improves overall SR from 38.45% to 40.69% and CR from 88.79% to 91.03%, yielding gains of 2.24 percentage points in both metrics. Improvements are observed at Levels II and III, while Level I shows a small decrease. These results indicate a modest overall benefit from the selected harness configuration, supporting execution experience as useful evidence for refining the model-external execution scaffold. This comparison complements the complete-system evaluation, which additionally includes feedback-guided recovery and persistent cross-question experience reuse.

D Examples of Acquired Harness Knowledge

This section presents representative knowledge units extracted from previous writable-memory runs. The accumulated knowledge primarily concerns EHR schema understanding, query construction, tool use, execution recovery, aggregation scope, and temporal reasoning. It should therefore be interpreted as execution-grounded EHR knowledge rather than general diagnostic or therapeutic knowledge.

1. **Patient-level counting.** Questions asking for the number of patients should generally use
   `COUNT(DISTINCT SUBJECT_ID)`.
   Counting joined rows directly can overestimate the result because one patient may have multiple admissions or clinical events.
2. **Admission-level joins.** Clinical event tables should normally be connected through `HADM_ID`. The Harness distinguishes the patient-level identifier `SUBJECT_ID` from the encounter-level identifier `HADM_ID`.
3. **Exact schema usage.** Generated programs must use table and column names exposed by the database tools. For example, MIMIC-III uses `prescriptions.DRUG`, not an invented field such as `DRUG_NAME`.

4. **Dataset-specific schemas.** The same clinical concept can be represented differently across MIMIC-III, eICU, and TREQS. The Harness therefore maintains dataset-specific table priorities and schema constraints.
5. **Case-insensitive value matching.** Text values may differ in capitalization. When supported, filters can use:

   ```
   WHERE DRUG = 'NOREPINEPHRINE' COLLATE NOCASE.
   ```

   This applies to values; table and column names must still match the exposed schema exactly.
6. **Population consistency.** The numerator and denominator of a percentage must be constructed from the same patient population. Admission, diagnosis, demographic, and temporal restrictions must be applied consistently.
7. **Temporal direction.** Terms such as *before*, *after*, and *during* imply different temporal constraints. The generated comparison must preserve the requested direction.
8. **Subtraction order.** For difference questions, the operand order described by the question must be preserved even if database values are retrieved in another order.
9. **Filtering before aggregation.** Minimum, maximum, average, percentage, and count operations should be performed only after applying all requested population, encounter, diagnosis, and time restrictions.
10. **Tool return types.** Tool outputs must be processed according to their actual types. A value returned by `GetValue(..., list)` is already a Python list and should not be treated as a DataFrame or split as a string.
11. **SQL result extraction.** `SQLInterpreter` may return a one-row tuple such as `[(42,)]`. When a scalar is requested, the correct row and column should be extracted before assigning `answer`.
12. **Localized runtime repair.** When execution fails, valid parts of the generated program should be retained. Only the failing column, filter, expression, join, or return-type operation should be repaired.
13. **Failure-specific recovery.** Different failures require different repair actions:
    - `bad_column`: inspect the exposed schema and replace the invalid field;
    - `syntax`: repair quotation marks, parentheses, or malformed SQL;
    - `return_type`: adapt processing to the actual tool output;
    - `logic`: restore missing variables, joins, or constraints;
    - `multi_value`: preserve multiple identifiers rather than forcing a scalar.
14. **Strict completion verification.** A task is complete only when the final execution succeeds and the assistant emits a standalone `TERMINATE`. An occurrence of `TERMINATE` in a prompt or an earlier attempt is insufficient.
15. **Current hospital admission.** A currently admitted patient is identified through:

    ```
    admissions.DISCHTIME IS NULL.
    ```

    This should not be confused with an ongoing ICU stay.
16. **Current ICU stay.** A current ICU stay is identified through:

    ```
    icustays.OUTTIME IS NULL.
    ```

    Hospital admission status and ICU status must be queried separately.
17. **Hospital versus ICU length of stay.** Hospital length of stay should use `admissions.ADMITTIME` and `admissions.DISCHTIME`. ICU length of stay should use the corresponding timestamps in `icustays`.
18. **Last completed admission.** To retrieve the latest completed hospitalization, admissions without a discharge timestamp should first be excluded. The remaining records are ordered by `ADMITTIME` in descending order.
19. **Patient anchoring before event retrieval.** Patient-specific questions should first identify the relevant patient and admission records. Prescriptions, diagnoses, laboratory events, and procedures should then be restricted to those admissions.

20. **Diagnosis-name resolution.** A diagnosis name should be resolved through:

$$\texttt{d_icd_diagnoses.SHORT_TITLE} \rightarrow \texttt{ICD9_CODE}.$$

The resulting code is then used to filter `diagnoses_icd`.

21. **Procedure-name resolution.** Procedure descriptions should be resolved through `d_icd_procedures` before filtering `procedures_icd`. A procedure description should not be compared directly with an ICD code column.
22. **Laboratory-item resolution.** Laboratory names should first be matched in `d_labitems`. The resulting `ITEMID` values are then used to retrieve measurements from `labevents`.
23. **Vital-sign resolution.** Vital-sign questions normally require resolving the requested measurement through `d_items` and retrieving timestamped observations from `chartevents`.
24. **Drug-route retrieval.** Medication route should be obtained from `prescriptions.ROUTE` after filtering `prescriptions.DRUG`. It should not be inferred through `d_items` or an input-event `ITEMID` loop.
25. **Medication scope within an encounter.** When a question refers to medication during the same hospital visit, prescription records must be restricted to the same `HADM_ID`.
26. **Multiple matching identifiers.** A dictionary lookup may return several valid ICD or item identifiers. All supported identifiers should be retained instead of selecting the first one without evidence.
27. **Empty identifier protection.** Before constructing a condition such as `HADM_ID IN (...)` or `ITEMID IN (...)`, the program should verify that the identifier collection is non-empty.
28. **List-valued outputs.** A value returned by `GetValue(..., list)` should be iterated directly. Calling `split()` on the resulting list is an incorrect type operation.
29. **Scalar versus collection outputs.** The agent must distinguish a scalar, list, tuple, and DataFrame. Attempting to access a list using DataFrame properties such as `columns` is a recurring failure.
30. **Preserving multi-valued answers.** Questions requesting multiple diagnoses, procedures, drugs, or measurements should retain the complete requested collection rather than silently reducing it to one value.
31. **Absolute-year filtering.** A fixed year such as "since 2104" should use the year of the relevant event timestamp:

```
strftime('%Y', EVENTTIME) >= '2104'.
```

32. **Relative-time filtering.** Expressions such as "last year" and "during the previous two years" require relative temporal modifiers. They should not be treated as fixed calendar-year conditions.
33. **Avoiding the machine clock.** MIMIC-III dates are shifted into synthetic future years. Calculations involving benchmark-relative current time should therefore use the database clock rather than Python's `datetime.now()`.
34. **Elapsed-time computation.** Elapsed days or hours should be computed from the two relevant database timestamps. A clinical timestamp should not be subtracted from the machine's wall-clock time.
35. **Temporal event ordering.** To determine whether event B occurred after event A, both events must first be associated with the requested patient and encounter before comparing their timestamps.
36. **Same-visit constraints.** "During the same hospital visit" requires equality of `HADM_ID`. Temporal proximity alone does not establish that two events occurred during the same encounter.
37. **Same-month constraints.** Events required to occur in the same month must match on both year and month. Comparing only the month number can combine events from different years.

38. **Strict versus inclusive boundaries.** The Harness distinguishes "before" and "after" from "on or before" and "in or after." The comparison operator must reflect the requested inclusiveness.
39. **Primary-diagnosis scope.** A question referring to a primary disease should not use every diagnosis recorded for an admission. The query should apply the dataset field or ordering rule that identifies the primary diagnosis.
40. **Discharge-destination filtering.** Discharge location is an admission-level attribute and should be filtered before aggregating patient age, diagnoses, or clinical events.
41. **Insurance and demographic restrictions.** Insurance, gender, ethnicity, language, and admission type must be applied to the same patient/admission population used by the downstream event query.
42. **Age aggregation.** Age extrema or averages should be computed only after all requested demographic, disease, insurance, mortality, and length-of-stay conditions have been applied:

    `MIN(AGE)`, `MAX(AGE)`, `AVG(AGE)`.
43. **Mortality-state filtering.** "Still alive" and "expired" should be mapped to the database's explicit mortality indicator rather than inferred from unrelated clinical events.
44. **TREQS schema boundary.** TREQS queries should use only its five exposed tables: `DEMOGRAPHIC`, `DIAGNOSES`, `PROCEDURES`, `PRESCRIPTIONS`, and `LAB`.
45. **TREQS table capitalization.** Stored failures showed that generated code sometimes used `demographic` when the execution environment exposed `DEMOGRAPHIC`. The repair rule preserves exact table names.
46. **TREQS event joins.** TREQS event tables should be joined to `DEMOGRAPHIC` through `HADM_ID`. Patient-count questions should aggregate distinct `DEMOGRAPHIC.SUBJECT_ID` values.
47. **TREQS medication routes.** Drug-route questions should filter `PRESCRIPTIONS.DRUG` and retrieve `PRESCRIPTIONS.ROUTE`. Route information should not be searched in diagnosis or procedure tables.
48. **SQL quotation repair.** A frequent TREQS failure was an unterminated SQL string caused by nested quotation marks. The stored repair pattern uses consistent outer and inner quoting or a multiline SQL string.
49. **Filter-language restrictions.** `FilterDB` supports a limited condition language. Complex joins, grouped aggregation, nested queries, or unsupported expressions should be executed through `SQLInterpreter`.
50. **Choosing the database tool.** `LoadDB`, `FilterDB`, and `GetValue` are suitable for simple single-table operations. `SQLInterpreter` is preferred for joins, distinct counts, grouped aggregation, and temporal comparisons.
51. **Undefined intermediate variables.** Repair code must preserve definitions of intermediate tables, identifiers, and filtered results. Replacing a final line while removing a required variable commonly produces a `NameError`.
52. **Index safety.** Before accessing the first or last element of a result, the program should verify that the collection is non-empty. This avoids repeated "list index out of range" failures.
53. **Type-safe string construction.** Identifiers and timestamp values should be converted explicitly when inserted into strings. Implicit concatenation between strings, integers, lists, and tuples is unsafe.
54. **Preserving working code during repair.** When only one operation fails, the remaining executable code should be retained. Restarting from scratch can introduce new errors into previously correct table selection and filtering logic.
55. **Final-answer assignment.** The final result must be assigned to the variable `answer`. Merely printing a value or leaving the result in an intermediate variable does not satisfy the agent execution contract.
56. **Stopping after successful execution.** After a valid result has been assigned, the agent should emit `TERMINATE` immediately. Continuing to revise a successful query can overwrite a correct answer or introduce another failure.

Persistent-memory statistics. Previous writable-memory runs produced 48 normalized pitfalls, 80 retained experiences, 64 reusable skills, and 120 recent execution states for the MIMIC-III setting. The TREQS Auto-Harness run retained 20 pitfalls, 80 experiences, 14 reusable skills, and 120 recent states. The state collection is a bounded recent-state buffer and should not be interpreted as 120 independent knowledge concepts.

Knowledge scope. The stored artifacts demonstrate learning of schema conventions, tool-use contracts, query patterns, temporal constraints, aggregation scope, and error-repair strategies. They do not independently establish the acquisition of new diagnostic, therapeutic, or clinical-guideline knowledge.

Knowledge provenance. The knowledge instances above are drawn from earlier writable-memory experiments and are included only to illustrate the types of operational knowledge that the harness can acquire. They are not used to define the memory-update protocol of the main `EHR-Evo` evaluation reported in Table 1.

E Devices

Local preprocessing, database execution, and evaluation were conducted on a Linux server equipped with an AMD EPYC 7742 64-Core Processor and NVIDIA A800-SXM4-80G GPUs. The language models used as the outer-loop proposer and frozen EHR solver were accessed through model services; no model training or parameter updates were performed in this work.

F AI Use Statement

We used an LLM for language polishing. All technical content and results were reviewed and verified by the authors. LLMs also serve experimental roles in the proposed system: GPT-5.6-sol is used as the outer-loop Harness proposer, while GPT-4-0613 serves as the frozen EHR solver for program generation and feedback-guided repair. These roles and model configurations are described in the paper and appendix.

G Ethics Statement

This work studies EHR question answering using existing de-identified clinical datasets and does not involve prospective patient recruitment or clinical deployment. Such data remain sensitive and must be handled under the applicable access and data-use requirements. Persistent memory may retain information from execution traces; deployment would therefore require controls on what is stored, retrieved, and shared with external model services.

Our evaluation measures benchmark answer accuracy and execution completion, not clinical safety or validity. Incorrect answers and accumulated memory errors could affect subsequent queries, so the system should not be treated as a validated clinical decision tool. Furthermore, benchmark-conditioned feedback is unavailable in routine clinical use. Real-world deployment would require independent validation, privacy safeguards, and appropriate human oversight.

H Reproducibility Statement

We provide detailed experimental settings in the paper and appendix, including dataset preprocessing, model configurations, memory initialization, retry policies, evaluation protocols, and relevant hyperparameters. We also provide an anonymized code repository containing the implementation and scripts required to reproduce the main experiments.

Reproduction additionally requires access to the corresponding EHR datasets and model services under their applicable terms.