CODECLINIC: EVALUATING AUTOMATION OF CODING SKILLS FOR CLINICAL REASONING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Clinical reasoning agents based on large language models (LLMs) aim to automate tasks such as intensive care unit (ICU) monitoring and patient state tracking from electronic health records (EHRs). Existing systems typically rely on manually curated clinical tools or skills for concepts such as sepsis detection and organ failure assessment. However, maintaining these tool libraries requires substantial expert effort, while zero-shot querying or code generation often produces inefficient and unreliable reasoning chains, especially under institution-specific clinical policies. We introduce CodeClinic, a benchmark built on MIMIC-IV for evaluating whether LLM agents can synthesize and compose reusable clinical skills instead of relying on fixed toolboxes. The benchmark contains two complementary tasks: longitudinal ICU surveillance and compositional information seeking. The longitudinal setting simulates monitoring patient trajectories with structured decisions every four hours across 25 findings and eight clinical families, while the compositional setting spans 63k instances across 259 tasks in nine domains and is stratified by compositional dependency depth to evaluate increasingly complex multi-step reasoning. We further propose an offline autoformalization pipeline that converts natural-language clinical guidelines into reusable and verified Python skill libraries through iterative LLM refinement. With the same backbone, the resulting library outperforms both zero-shot code generation and a published expert-engineered EHR toolbox while using up to 47% fewer inference tokens per query, yet leaves substantial headroom on both tasks.[1]

1 INTRODUCTION

Large language models (LLMs) have demonstrated remarkable potential for clinical reasoning, achieving expert-level performance on medical licensing examinations (Singhal et al., 2023) and strong capabilities across diverse medical tasks (Bedi et al., 2026; Wu et al., 2025; Wang et al., 2025). Recent works extend these capabilities to agentic settings, where LLMs generate code or queries against electronic health record (EHR) databases and reason over the results to answer clinically meaningful questions (Liao et al., 2026; Tang et al., 2024).

A common approach equips agents with a user-designed toolbox of domain-specific functions for recurring clinical computations. Toolboxes improve reliability but do not scale well: designing and maintaining them requires sustained expert effort, and their coverage is limited by the author's foresight. The alternative, a generic interface such as a single `query_db` function, can hurt both accuracy and cost, as the agent must repeatedly issue complex queries and re-derive intermediate results. Moreover, many clinical concepts, such as sepsis screening, organ failure scoring, and vasopressor thresholds, are not universal: their precise definitions vary across institutions and evolve with guidelines, so an agent without access to the relevant policies faces an underspecified problem.

Our benchmark therefore targets a central question: *Given limited verification data, how well can LLMs synthesize modular, composable libraries that operate over EHR data while accounting for institutional definitions and policies?* We introduce CodeClinic, a benchmark built on MIMIC-IV (Johnson et al., 2020) with two complementary tasks. The first is longitudinal Intensive Care Unit

[1]Code to reproduce data and evaluation: `https://anonymous.4open.science/r/CodeClinic-CE38`

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.

(ICU) surveillance: an agent issues structured monitoring decisions at 4-hour checkpoints over a 13-step horizon, covering 25 findings across eight clinical families, which requires persistent state tracking, handling time-sensitive variables, and recomputing composite conditions (e.g., sepsis) without access to future information. It is a *current-state* surveillance task (inferring the present state from evidence up to each checkpoint), not a forecasting task. The second is compositional information seeking, spanning 63,000 queries over 63 clinical concepts in nine domains, stratified by dependency depth to measure multi-step reasoning.

As a baseline, we propose an offline autoformalization pipeline that converts natural-language guideline summaries into a reusable, verified Python function library via iterative LLM refinement. The library is constructed once with limited supervision and reused across queries. With the same backbone, it outperforms a published expert-engineered EHR toolbox (Liao et al., 2026) by 15.5 points while using 47% fewer inference tokens, although the expert toolbox remains stronger on irregular temporal retrieval. We scope our empirical claims to reuse across tasks, concepts, and dependency depths within MIMIC-IV, with initial evidence of transfer to a second database (eICU) (Appendix J). Our contributions are:

- **CodeClinic (Benchmark)**: a longitudinal agentic ICU monitoring task paired with a compositional information seeking task, with a detailed label policy and leakage-tested evaluation harness.
- **Benchmark Analysis (Findings)**: a systematic characterization of where and why current LLMs fail, isolating long-horizon state maintenance as the dominant longitudinal failure and multi-hop composition as the dominant compositional failure.
- **Clinical Autoformalization (Baseline)**: a pipeline that builds a verified function library from limited labels, outperforming zero-shot code generation and an expert-engineered toolbox, with full offline cost accounting and break-even analysis.

2 Related Work

EHR Databases and Benchmarks. Most prior EHR benchmarks build on the MIMIC family (Johnson et al., 2020; 2018). Text-to-SQL efforts (MIMICSQL (Wang et al., 2020), EHRSQL (Lee et al., 2022), EHR-SeqSQL (Ryu et al., 2024)) and broader QA benchmarks (DrugEHRQA (Bardhan et al., 2022), EHRXQA (Bae et al., 2023), EHRNoteQA (Kweon et al., 2024)) cover constrained query spaces over structured tables, multimodal records, or notes. EHRSHOT (Wornow et al., 2023) adds point-in-time foundation-model prediction, and code-generation benchmarks (EHRAgent (Shi et al., 2024), EHRStruct (Yang et al., 2026)) frame EHR reasoning as iterative code generation. All evaluate static, individually posed queries with expert-curated toolboxes, and none jointly stratify multi-hop concept dependencies and time-evolving clinical state. CodeClinic is the only benchmark we know of that combines compositional dependency-depth stratification with longitudinal reasoning over full ICU trajectories (Table 5).

LLM-Based Medical Agents. A growing body of work builds LLM agents that interact with EHR data or simulated clinical environments (Liao et al., 2026; Tang et al., 2024; Hager et al., 2024; Bani-Harouni et al., 2025; Xu et al., 2025; Huang et al., 2025; Jiang et al., 2025; Schmidgall et al., 2024), with broader medical evaluation suites such as MedHELM (Bedi et al., 2026) measuring static task performance. Two limitations recur: agents rely on hand-crafted tool environments whose completeness bounds the evaluation, and re-executing full tool chains per query incurs significant token overhead. Our work is orthogonal: instead of evaluating an agent inside a fixed tool environment, we ask how well LLMs can *generate* that environment.

Autoformalization and Code-Augmented Reasoning. Autoformalization translates informal statements into executable form, an approach originally developed for mathematics by translating theorems into Lean or Isabelle (Wu et al., 2022). We adapt the idea to clinical guidelines, replacing proof checkers with clinical verification and targeting a Python function library. Because clinical definitions vary across institutions, the loop grounds general guideline summaries in local practice by verifying generated code against labeled cases from the target EHR system. Our pipeline runs a ReACT-style (Yao et al., 2023) code-as-reasoning loop (Gao et al., 2023; Chen et al., 2021) *offline* to produce a stable, reusable library, separating costly library construction from inference and avoiding the repeated token expenditure of agents that re-derive the same computations at query time.

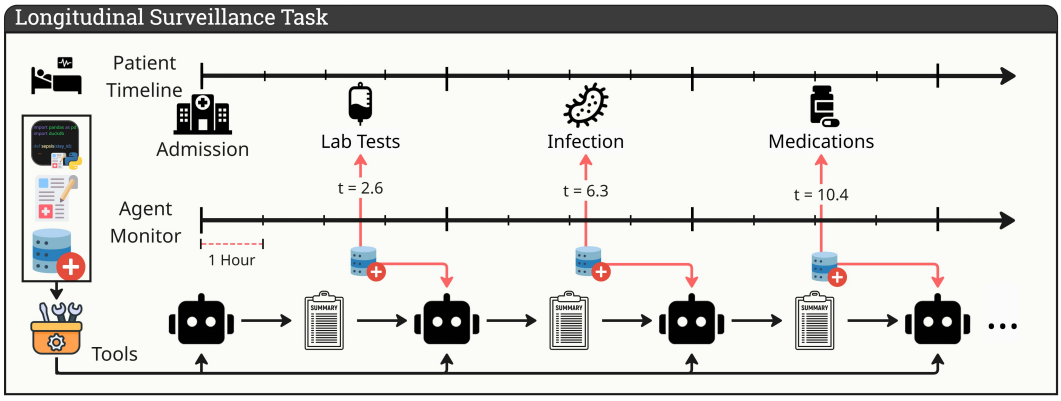


Figure 1: Illustration of the longitudinal ICU surveillance task in CodeClinic. The agent is prompted every 4 hours throughout an ICU stay to assess the patient's current condition, with strict visibility constraints that limit it to data available up to the current checkpoint. At each checkpoint, the agent either reasons or writes code to gather evidence and outputs a surveillance decision. A rolling history of prior checkpoint decisions is carried forward across the stay via a summarization step.

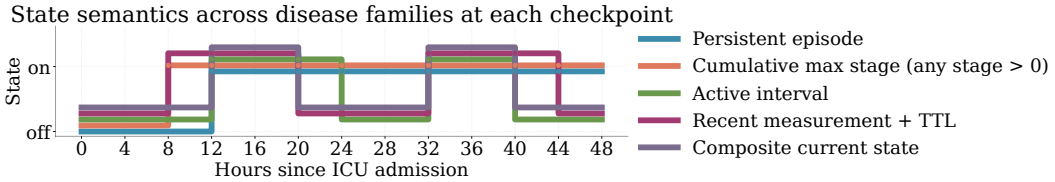


Figure 2: Example state dynamics across an ICU trajectory for different medical concepts. CodeClinic challenges models to interpret these changes in state dynamics across the patient timeline.

3 BENCHMARK

CodeClinic comprises two tasks over MIMIC-IV (Johnson et al., 2023): *longitudinal ICU surveillance* (Figure 1), which stresses state tracking across rolling checkpoints, and *compositional information seeking* (Figure 3), which stresses multi-hop reasoning over the dependency graph of clinical concepts. CodeClinic is the only EHR benchmark we know of that combines both properties (Table 5, Appendix C).

3.1 LONGITUDINAL ICU SURVEILLANCE

We draw a cohort of roughly 46k ICU stays from MIMIC-IV by keeping only stays with at least 48 hours of in-unit time, so that every trajectory is long enough to expose meaningful clinical evolution. Each stay is sliced into a fixed sequence of decision points spaced every four hours from admission through hour 48, giving 13 checkpoints per stay. Patients are assigned to splits at the subject level, so no patient appears in both the train and test set. From this cohort we release a held-out 2,000 stay benchmark, sampled with a stratification which preserves the natural mix of ICU patients while preserving rarer high-acuity states.

Task Setup At each checkpoint, the agent emits a structured decision: a set of *suspected conditions*, a set of *alerts* that have crossed escalation thresholds, a single *global action* (continue monitoring or escalate care), and a coarse *priority* level. This interface avoids committing to a fixed disease ontology while requiring the model to reason over the full multi-organ state at every step. In the source cohort, over 90% of stays trigger at least one surveillance family within the first day and roughly two-thirds trigger three or more (statistics in Appendix D).

Label Generation We maintain a registry of 25 ICU surveillance findings in eight clinical families covering the dominant ICU monitoring concerns (Table 1). For each finding we define a deterministic builder over MIMIC-IV that consumes the patient's data up to the current checkpoint and decides

Table 1: The 25 ICU surveillance findings by clinical family and temporal-semantics class. A single memory heuristic cannot satisfy all families simultaneously, which is the core source of longitudinal difficulty (Figure 2). Classes: persistent episode, cumulative max stage, active interval, recent-measurement TTL, composite current state. Full per-finding triggers, thresholds, windows, and source builders are in Appendix D, Table 8.

Family	#	Temporal class(es)	Representative findings (threshold)
Infection	2	persistent episode	infection suspected / culture-confirmed
Sepsis	1	persistent episode	Sepsis-3 onset
Renal	6	cumulative max / TTL / interval	AKI KDIGO 1–3; oliguria; CRRT
Respiratory	4	active interval / TTL	invasive vent.; PF ratio $< 200, < 100$
Hemodynamic	4	active interval / composite	vasoactive support; septic shock (lactate $\geq 2, 4$)
Neurologic	2	recent-measurement TTL	GCS 9–12; GCS ≤ 8
Metabolic	4	recent-measurement TTL	lactate $\geq 2, 4$; pH $< 7.30, \leq 7.20$
Coagulation	2	recent-measurement TTL	INR $\geq 1.5, \geq 2.0$

whether the finding is currently active. The builder for each family encodes the appropriate clinical *temporal semantics* (Figure 2). Monotonic syndromes such as sepsis remain active once triggered, whereas interventions such as ventilation are active only while in progress. A model therefore cannot remember a single binary label per disease. It must track which families persist, which decay when evidence becomes stale, and which are recomputed from evolving components.

Finding Selection and Sourcing We prioritize findings that are clinically meaningful monitoring states, matter for escalation or follow-up, are definable from the MIMIC-Code derived schema, and are frequent enough in MIMIC-IV to support evaluation, while enriching rarer high-acuity states (Acute Kidney Injury (AKI), septic shock, etc). The complete per-finding specification (trigger, temporal class, threshold, window, and source builder) is in Table 8. Concept values and clinical thresholds follow the community-validated MIMIC-Code pipeline and its published scores.

Agent Loop At each checkpoint the agent sees the current hour, a summary of its own previous decisions and observations, and a toolbox for querying the database. Tools can expose only the raw EHR tables or, additionally, a function library such as the one produced by our pipeline (Section 4), allowing us to measure the contribution of higher-level clinical tooling. The agent retrieves evidence, commits to a decision, and writes a short summary visible at the next step. To prevent temporal leakage, at each checkpoint the database is exposed through time-bounded views that filter every source table to rows observed at or before the checkpoint time, so the agent cannot query future measurements. Derived tables are excluded from the accessible schema for both tasks. In early experiments with them exposed, agents simply looked up the answer instead of implementing clinical logic, measuring retrieval rather than reasoning.

3.2 MIMIC-IV Compositional Information Seeking

All questions are derived from the MIMIC-IV `derived` schema, structured representations of the raw ICU data produced by community-validated SQL transformations (Johnson et al., 2021). The benchmark spans 63 concepts in nine domains: comorbidity scoring, demographics, first-day assessments, laboratory measurements, medications, organ failure indices, severity scores, sepsis-related variables, and ICU treatments.

Concept Definition Each concept corresponds to one derived table (e.g., `sofa`) and specifies (i) a base SQL query joining it with ICU stays and admissions to produce a per-stay feature matrix, and (ii) question variants, each targeting a clinical attribute and question type. For each concept we also provide an *LLM-generated guideline summary* (clinical criteria, measurements, and time windows) synthesized from retrieved PubMed literature (Appendix E). These summaries are neither verbatim clinical practice guidelines nor institutional policy, and they are not the source of ground truth. Instead, they are a prior supplied to the agent, so drift from the literature can add noise to the agent's input but cannot corrupt the benchmark labels.

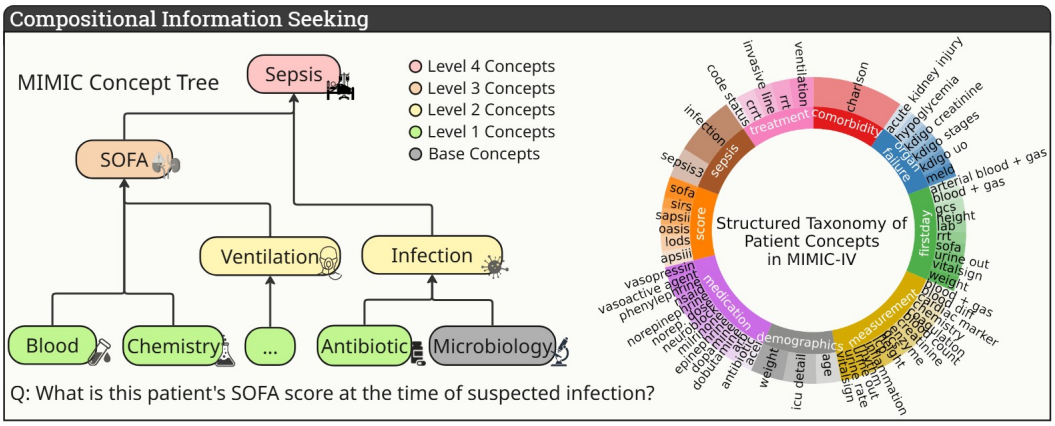


Figure 3: **Left:** Example sepsis-related question with its corresponding dependency graph. Answering the query requires composing intermediate clinical concepts (e.g., Sequential Organ Failure Assessment (SOFA) score used in organ dysfunction definitions for sepsis, suspected infection) that must be recomputed over time. **Right:** Hierarchical taxonomy of MIMIC clinical concepts, organized by supercategory and associated sub-concepts.

Dependency-Based Difficulty A key property of MIMIC-IV derived tables is that many are computed from other derived tables, inducing a directed acyclic dependency graph. We assign each concept a difficulty level equal to its longest dependency-chain depth in this graph. Concepts with no upstream dependencies (e.g., vitalsign, norepinephrine) are Level 1, representing raw chart or medication extractions. More challenging concepts such as ventilation or creatinine_baseline are Level 2, requiring one layer of aggregation. Concepts at depth $\geq$3 (e.g., sofa, sepsis3) are grouped as Level 3+, reflecting composite clinical constructs that integrate multiple subsystems. The graph is built by parsing SQL definitions in the MIMIC-Code repository and expanding with conditions derived from ICD-10 diagnoses and UMLS relations (Appendix F); Figure 3 shows an example.

Sampling and Splits For each concept, up to 1000 ICU stays are sampled in total, distributed across its question variants. A 10% subset (100 rows per variant) forms the training split used during autoformalization. We use a small training split to simulate a realistic scenario where labeled verification data is scarce and expensive. The remaining 90% is the held-out test set. Applying this process results in 63,000 compositional queries across 42,311 unique ICU stays (6,300 train / 56,700 test). Ground-truth labels are extracted deterministically from derived tables, and answers are scored with deterministic parsers rather than an LLM judge, which would dominate evaluation cost at this scale. Unparseable outputs fall back to the majority label to estimate a conservative lower bound.

4 Baseline: Clinical Autoformalization

We propose a clinical autoformalization pipeline (Figure 4) that converts natural-language clinical guideline summaries into a tested, reusable Python function library using only a small train split as supervision. The pipeline takes as input a verification dataset $\mathcal{Q} = \{(q_k, y_k)\}_{k=1}^{K}$, an EHR database $\mathcal{C}$, and LLM-generated guideline summaries $\mathcal{G}$, and produces a verified Python function f that deterministically extracts a clinical concept for any patient. The full algorithm is in Appendix G.

Tool generation. The core of the pipeline is an iterative ReACT loop (Yao et al., 2023) in which an LLM agent explores the EHR database schema, consults guideline summaries, and writes a Python function for a target concept. The agent operates within a stateful code interpreter exposing three tools: `query_db(sql)` for database access, `search_guidelines`/`get_guideline` for retrieving clinical specifications from $\mathcal{G}$, and `search_functions`/`load_function` for discovering and importing previously verified functions. Guideline summaries are exposed as a searchable library rather than injected into the prompt, so the agent must bridge clinical knowledge and database schema through exploration.

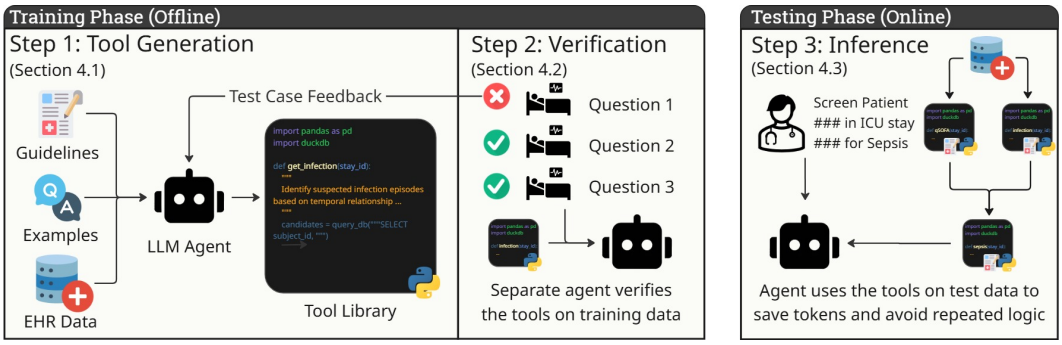
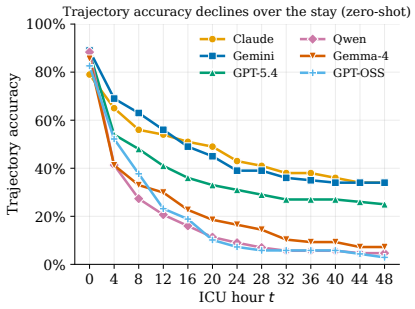


Figure 4: Overview of the autoformalization pipeline. During the offline *training phase*, an LLM agent uses medical guidelines, verification examples, and EHR data to construct a library of executable tools. A separate verification agent evaluates this library on the remaining training data, with errors fed back in an iterative refinement loop. During the online *testing phase*, the learned tool library is provided to the inference agent, enabling more efficient computation and reduced token usage.



Model	Acc.	Traj. Acc.	Prio.	Alerts F1	Susp. F1
Frontier Models[1]					
Claude-Sonnet-4.6	**76.2**	**34.0**	56.8	**47.2**	38.3
Gemini-3.1-Pro	72.5	**34.0**	**59.4**	46.4	**43.9**
GPT-5.4	66.5	25.0	51.7	32.1	25.6
Open-Source Models					
Qwen3.5-4B	27.0	5.0	24.7	24.0	18.3
Qwen3.5-9B	24.9	6.0	24.1	24.1	18.5
Qwen3.5-27B	31.8	3.0	31.1	21.9	20.6
Gemma-4-31B	55.1	**7.2**	38.1	**26.5**	20.0
GPT-OSS-120B	**55.3**	2.9	**43.8**	21.9	**27.5**

Figure 5: **Left:** Trajectory accuracy on the rolling ICU surveillance task declines as monitoring windows lengthen. **Right:** Longitudinal monitoring metrics (%) with the zero-shot base-table interface. Even frontier models degrade from high per-checkpoint accuracy to low trajectory accuracy (Claude 76.2 $\rightarrow$ 34.0), reflecting the difficulty of maintaining a consistent multi-family state over the full stay.

Verification. Once the agent produces a candidate function f_t, it is evaluated against the 10% train split of $\mathcal{Q}$. A separate agent calls f_t in its own code interpreter and parses a verdict for each question to compare against ground truth. If accuracy exceeds $\theta = 0.90$, the function is accepted and saved to the shared library. Otherwise, per-question error traces are fed back to the autoformalization agent as structured feedback for the next iteration. The best-performing candidate across all iterations is retained; if no candidate reaches θ within the budget, it enters the library as a best-effort function (Section 5.2).

Inference. Once the library $\mathcal{L} = \{f_1, \ldots, f_M\}$ is complete, it is deployed as a static resource over the held-out 90% test and other clinical tasks. An LLM inference agent receives a question q along with the available function signatures in $\mathcal{L}$ and access to the database $\mathcal{C}$, and iteratively calls the functions it deems relevant, reasoning over their outputs to produce an answer $\hat{y} = \text{LLM}(q, \mathcal{L}, \mathcal{C})$. Because the functions in $\mathcal{L}$ have been verified on labeled cases, the inference agent's context is occupied by compact function signatures rather than raw schema exploration, substantially reducing token usage. The same function produces identical output for the same patient across runs, which is critical for reproducibility in clinical settings.

5 Results

Baselines. Each agent is allowed up to 15 conversation turns of exploration (10 for the longitudinal task) before being prompted for a final answer; an example trajectory is given in Appendix L.

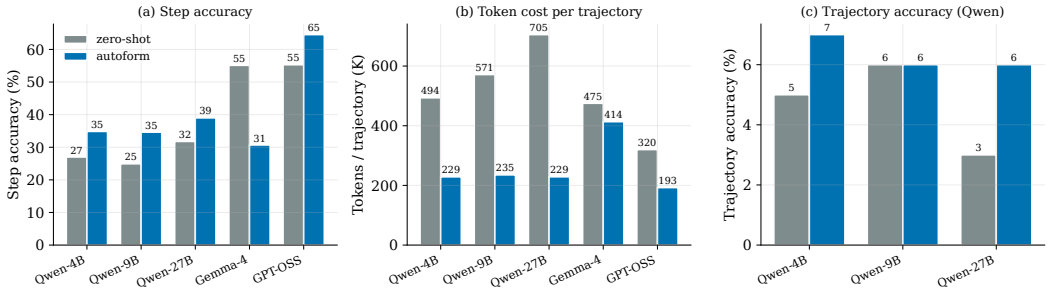


Figure 6: Zero-shot vs. autoformalization on the longitudinal task (open-weight models). Figure **(a)** shows per-checkpoint step accuracy, figure **(b)** shows tokens per trajectory (all five models), and figure **(c)** displays trajectory accuracy for Qwen models. Autoformalization raises step accuracy and cuts token cost, but trajectory accuracy is essentially unchanged.

(1) Zero-shot: the LLM receives only the raw MIMIC-IV schema and a `query_db` function.
(2) GRPO finetuning: the zero-shot interface, with `Qwen3.5-4B` (Qwen Team, 2026) finetuned via GRPO (Shao et al., 2024) on 50 training examples per concept, rewarding exact match with the ground truth (Appendix I).
(3) Clinical Autoformalization (ours): the raw schema, `query_db`, and the autoformalized library $\mathcal{L}$ generated from the compositional train split. We use the same library, built with `Qwen3.5-27B`, across all runs.

Evaluation Metrics. For the **longitudinal task**, each checkpoint is scored on global-action accuracy (**Acc**), priority accuracy (**Prio**), and macro set-based F_1 over the predicted suspected-conditions and alerts sets (**Susp. F1**, **Alerts F1**). **Traj. Acc** is the fraction of stays where the global action is correct at *all* 13 checkpoints, so a single error anywhere in the horizon counts as a failure. For the **information seeking task**, answers are judged by exact match with 1% tolerance, then macro-averaged across variants and concepts. **Tokens/Q** counts all tokens (prompt + generated) across all agent turns per question.

5.1 Longitudinal Monitoring Results

Figure 5 reports surveillance metrics and global-action accuracy as the trajectory unfolds. We found that the GRPO finetuned model (Section 5.2) did not generalize to this task, so we limit the analysis to zero-shot and autoformalization.

Agents struggle to track patient state dynamics. Even the strongest model (Claude-Sonnet-4.6) reaches only 34.0% trajectory accuracy despite 76.2% checkpoint accuracy. Because monotonic syndromes persist while other states decay (Figure 2), models that track only the latest checkpoint or only the running maximum are penalized by both regimes.

Disease tracking is harder than predicting actions. Across all models, global-action accuracy ($\sim$ 25–76%) is much higher than suspected-conditions F_1 ($\sim$ 18–44%) and alerts F_1 ($\sim$ 22–47%). Producing the *full multi-organ picture*, rather than only the dominant problem, remains hard.

Autoformalized functions lower per-step difficulty, but longitudinal consistency remains hard. As shown in Figure 6, most models equipped with the autoformalized library achieve higher step accuracy than their zero-shot counterparts, with lower token cost across all models. The agent re-derives fewer evidence pathways from the raw schema at each checkpoint, but these local gains do not compound into trajectory-level improvement: trajectory accuracy for the Qwen family stays at its zero-shot level (Figure 6c).

Error analysis: the dominant error mode is temporal state tracking With the autoformalized library, tool calls succeed on 98.7–100% of checkpoints, so failures are not explained by schema navigation or code execution. Instead, agents collapse to conservative omission, predicting

[1]No data was saved or used for training by these model APIs, in compliance with the MIMIC-IV data use agreement. Frontier models were run on a 21% stratified subset of the data due to cost.

Model	Accuracy by Difficulty Level				Overall	Tokens/Q
	Level 1	Level 2	Level 3+	Δ (L1→L3+)		
Frontier Models[1]						
Claude-Sonnet-4.6	**58.4**	51.0	**41.1**	-17.3	**53.1**	80,308
Gemini-Pro-3.1	58.0	**54.1**	37.2	-20.8	53.0	**15,897**
GPT-5.4	55.1	47.8	36.7	-18.4	49.6	16,778
Open-Source Models						
Gemma4-31B	**46.9**	**44.2**	**32.2**	-14.7	**43.4**	39,243
gpt-oss-120B	39.4	38.6	23.2	-16.2	36.1	**19,576**
Qwen3.5-27B	36.5	38.6	21.6	-14.9	34.2	34,414
Qwen3.5-9B	22.8	22.9	8.7	-14.2	20.2	41,378
Qwen3.5-4B	25.4	20.9	8.6	-16.8	21.0	49,445

Table 2: Zero-shot accuracy (%) on the compositional information seeking test set by difficulty level. All models are given only the raw MIMIC-IV schema and a `query_db` tool.

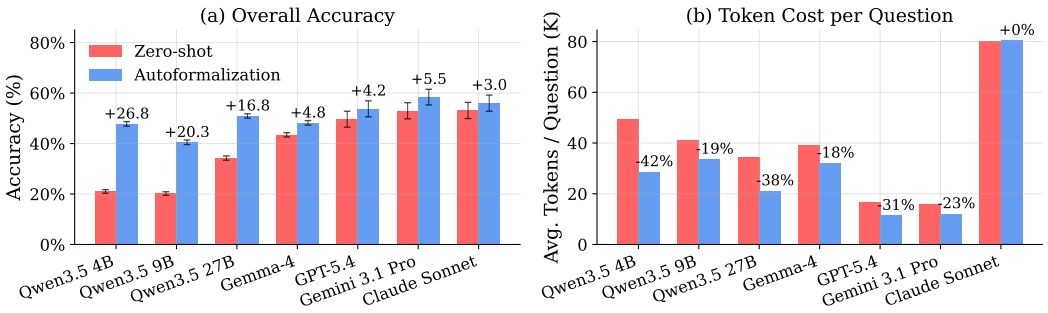


Figure 7: Zero-shot evaluation vs. our autoformalization baseline on the compositional information seeking task across model families. Autoformalization improves accuracy while improving or matching token efficiency for all models, with the largest relative benefit for smaller models that frequently fail to chain dependent clinical computations. Error bars: 95% bootstrapped intervals over 2,000 resamples.

`continue_monitoring` on 84–88% of checkpoints and an empty finding set on 77–86%. Errors are primarily misses: recall is near zero across all temporal-semantics classes, and models drop essentially all persistent states that should carry forward while over-persisting stale evidence in only 4–9% of cases. Consequently, global-action accuracy decays from 90% to 30% over the course of a 48 hour stay as clinical states accumulate. Gemma-4-31B's step-level regression under autoformalization (-24 points) is driven by this conservative collapse, not tool errors. Autoformalization thus isolates this core difficulty of long-horizon maintenance of persistent and cumulative findings.

5.2 Compositional Information Seeking Results

Task difficulty. Table 2 evaluates models spanning several families and scales (to control for schema memorization, as MIMIC-IV is widely studied) with the zero-shot interface. The best model reaches only 53.1%, and accuracy declines from Level 1 to Level 3+ for every model, reflecting compounding multi-hop dependencies (e.g., Sepsis-3 requires SOFA sub-scores, infection criteria, and antibiotic timing, each of which can fail).

Autoformalization vs. an expert-engineered toolbox. On this task, we additionally compare against the hand-curated, published AgentEHR tool set for MIMIC-IV (Liao et al., 2026) (table/schema navigation, time-windowed retrieval, keyword/value lookup), given alongside the raw schema. AgentEHR provides expert-designed *retrieval primitives*, not expert implementations of our 63 concepts. The closest concept-level alternative, wrapping MIMIC-Code SQL as tools, is degenerate because those queries produce the labels themselves. To our knowledge, no independently engineered implementa-

Method (Qwen3.5-27B)	Accuracy [95% CI]	Tokens/Q
Zero-shot code generation	30.6 [29.5, 31.8]	34,098
Expert toolbox (AgentEHR)	34.0 [32.8, 35.2]	37,902
Autoformalization (ours)	**49.5** [48.3, 50.8]	**20,234**

Table 3: Toolbox comparison with a shared backbone on the same 63 concepts and 6,300 questions (100 per concept, sampled once).

tion of these concepts is public. Table 3 compares the three toolboxes under a shared `Qwen3.5-27B` backbone. Autoformalization outperforms the expert toolbox by $+15.5$ points (non-overlapping CIs) and zero-shot code generation by $+18.9$, while using 46.6% and 40.7% fewer inference tokens, respectively. The gap is widest at the deepest dependency level (Level 3+: 42.8% vs. 23.8% for the expert toolbox), where questions compose several derived concepts. However, the expert toolbox is not universally worse. It outperforms on 11 of 63 concepts, with the largest margins on concepts dominated by irregular time-series retrieval.

Error analysis by question type. The results do not support a single retrieval-versus-composition explanation. Comparative questions are hardest for the generic baselines (zero-shot 14.4%, AgentEHR 14.9%), whereas autoformalization reaches 46.0%, indicating that reusable generated functions substantially improve multi-concept composition. Conversely, AgentEHR outperforms autoformalization on temporal questions (41.7% vs. 37.8%), so flexible temporal retrieval remains a specific, reproducible weakness of our method.

Full cost accounting. Building the 63-concept library, including all repair iterations and rejected candidates, consumed about 412.2M tokens (82.9M for generation and repair, 329.3M for verification) over 31.3K LLM calls ($\sim$88 GPU-hours). Dividing this by the per-question savings (13,864 vs. zero-shot; 17,668 vs. AgentEHR) gives a break-even point of 29.7K questions (470 per concept) against zero-shot and 23.3K (370 per concept) against the expert toolbox, which is less than the test split of our benchmark.

Method (Qwen3.5-4B)	Accuracy by Level			Overall	Tokens/Q
	Level 1	Level 2	Level 3+		
Zero-shot	44.9	32.8	26.3	38.1	30,479
GRPO finetuning	**57.1**	36.5	27.9	40.5	**22,376**
Autoformalization (ours)	51.2	**41.5**	**34.1**	**45.3**	23,569

Table 4: Weight adaptation (GRPO) vs. library learning on the held-out compositional test set. Accuracy (%) is macro-averaged across concepts; token usage is per question at inference.

Library learning vs. weight adaptation. Table 4 compares GRPO finetuning against autoformalization under a shared `Qwen3.5-4B` backbone, isolating the effect of learning reusable code from that of adapting model weights. Finetuning gives the best Level 1 accuracy (57.1%), likely by teaching shallow database search patterns, but its gains shrink on Level 2 and 3+ tasks that require deeper composition. Autoformalization achieves the best overall accuracy (45.3% vs. 40.5% for GRPO and 38.1% zero-shot) without parameter updates and with 23% fewer tokens than zero-shot, and its gains hold across model families (Figure 7).

6 Conclusion

We presented CodeClinic, an agentic benchmark consisting of 63k coding instances spanning 259 tasks over 63 clinical concepts, designed to evaluate whether LLMs can generate modular, composable libraries for EHR data processing. As a strong baseline, we proposed clinical autoformalization, which converts natural-language clinical guidelines into verified Python libraries via a QA-driven refinement loop, removing the need for manual tool engineering. The resulting libraries also generalize to the longitudinal task and reduce per-query token usage compared to zeroshot approaches. We see opportunity in extending CodeClinic to multimodal data sources and in using the verified libraries for deploying smaller clinical agents for realistic ICU monitoring workflows.

Ethics Statement

This work uses MIMIC-IV, a de-identified credentialed-access dataset distributed through PhysioNet. We release no patient data: users must hold their own PhysioNet credentials and build the databases locally. The framework supports fully local execution with open-weight models served via vLLM, which we document as the default so that no patient-derived content leaves the user's environment. Frontier-model experiments used API settings under which no data was retained or used for training, and hosted-API use by others is their responsibility under PhysioNet's guidance on responsible LLM use with credentialed data. CodeClinic is a research benchmark; neither its labels nor the autoformalized functions have been clinically validated, and they should not be used for patient care without institution-specific clinical review (Section 6). **Broader impact.** Verified, reusable clinical code is more auditable than one-off model outputs, but errors in generated functions could propagate silently into decision support, and MIMIC-IV's demographic skew may reduce reliability for underrepresented patient groups.

Reproducibility Statement

Code for building the benchmark from a local MIMIC-IV installation, running all agents, and scoring is available at `https://anonymous.4open.science/r/CodeClinic-CE38`, with pinned requirements and command-line arguments for all paths. A pytest suite runs end to end on the public MIMIC-IV Demo, including temporal-leakage tests (every row returned at a checkpoint is timestamped at or before it), tests that no agent query touches label-source derived tables, and parser/scoring tests. Benchmark construction is described in Section 3.1 and Appendices D–F; the autoformalization algorithm in Appendix G; all prompts in Appendix H; GRPO hyperparameters in Appendix I; and compute in Appendix K.

AI Use Statement

We used generative AI tools to assist with drafting and editing sections of the paper and with reformatting it for this venue. LLMs are also a subject of study in this work (as evaluated agents, and to generate guideline summaries as described in Appendix E). All authors reviewed AI-assisted text for accuracy, and we take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

Seongsu Bae, Daeun Kyung, Jaehee Ryu, Eunbyeol Cho, Gyubok Lee, Sunjun Kweon, Jungwoo Oh, Lei Ji, Eric Chang, Tackeun Kim, and Edward Choi. EHRXQA: A multi-modal question answering dataset for electronic health records with chest x-ray images. In *Advances in Neural Information Processing Systems*, 2023.

David Bani-Harouni, Chantal Pellegrini, Ege Özsoy, Matthias Keicher, and Nassir Navab. Language agents for hypothesis-driven clinical decision making with reinforcement learning. *arXiv preprint arXiv:2506.13474*, 2025.

Jayetri Bardhan, Anthony Colas, Kirk Roberts, and Daisy Zhe Wang. DrugEHRQA: A question answering dataset on structured and unstructured electronic health records for medicine related queries. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Jan Odijk, and Stelios Piperidis (eds.), *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pp. 1083–1097, Marseille, France, June 2022. European Language Resources Association. URL `https://aclanthology.org/2022.lrec-1.117/`.

Suhana Bedi, Hejie Cui, Miguel Fuentes, Alyssa Unell, Michael Wornow, Juan M Banda, Nikesh Kotecha, Timothy Keyes, Yifan Mai, Mert Oez, et al. Holistic evaluation of large language models for medical tasks with medhelm. *Nature Medicine*, pp. 1–9, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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.

Paul Hager, Friederike Jungmann, Robbie Holland, Kunal Bhagat, Inga Hubrecht, Manuel Knauer, Jakob Vielhauer, Marcus Makowski, Rickmer Braren, Georgios Kaissis, et al. Evaluation and mitigation of the limitations of large language models in clinical decision-making. *Nature medicine*, 30(9):2613–2622, 2024.

Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, et al. Biomni: A general-purpose biomedical ai agent. *biorxiv*, pp. 2025–05, 2025.

Yixing Jiang, Kameron C. Black, Gloria Geng, Danny Park, James Zou, Andrew Y. Ng, and Jonathan H. Chen. MedAgentBench: A virtual EHR environment to benchmark medical LLM agents. *NEJM AI*, 2(9), 2025. doi: 10.1056/AIdbp2500144.

Alistair Johnson, Lucas Bulgarelli, Tom Pollard, Steven Horng, Leo Anthony Celi, and Roger Mark. Mimic-iv. *PhysioNet. Available online at: https://physionet. org/content/mimiciv/1.0/(accessed August 23, 2021)*, 2020.

Alistair Johnson, Lucas Bulgarelli, Tom Pollard, Steven Horng, Leo Anthony Celi, and Roger Mark. MIMIC-IV (version 1.0). *PhysioNet*, 2021.

Alistair EW Johnson, David J Stone, Leo A Celi, and Tom J Pollard. The mimic code repository: enabling reproducibility in critical care research. *Journal of the American Medical Informatics Association*, 25(1):32–39, 2018.

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

Sunjun Kweon, Jiyoun Kim, Heeyoung Kwak, Dongchul Cha, Hangyul Yoon, Kwanghyun Kim, Jeewon Yang, Seunghyun Won, and Edward Choi. EHRNoteQA: An LLM benchmark for real-world clinical practice using discharge summaries. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3958.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Cheol Kim, and Edward Choi. EHRSQL: A practical text-to-SQL benchmark for electronic health records. In *Advances in Neural Information Processing Systems*, 2022.

Yusheng Liao, Chuan Xuan, Yutong Cai, Lina Yang, Zhe Chen, Yanfeng Wang, and Yu Wang. Agentehr: Advancing autonomous clinical decision-making via retrospective summarization. *arXiv preprint arXiv:2601.13918*, 2026.

Qwen Team. Qwen3.5. `https://huggingface.co/Qwen`, 2026. Qwen3.5 model family.

Jaehee Ryu, Seonhee Cho, Gyubok Lee, and Edward Choi. EHR-SeqSQL: A sequential text-to-SQL dataset for interactively exploring electronic health records. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 16388–16407. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.971. URL `https://aclanthology.org/2024.findings-acl.971/`.

Samuel Schmidgall, Rojin Ziaei, Carl Harris, Eduardo Reis, Jeffrey Jopling, and Michael Moor. AgentClinic: A multimodal agent benchmark to evaluate AI in simulated clinical environments. *arXiv preprint arXiv:2405.07960*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.1245. URL https://aclanthology.org/2024.emnlp-main.1245/.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180, 2023.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 599–621, 2024.

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*, WWW '20, pp. 350–361, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370233. doi: 10.1145/3366423.3380120. URL https://doi.org/10.1145/3366423.3380120.

Shansong Wang, Mingzhe Hu, Qiang Li, Mojtaba Safari, and Xiaofeng Yang. Capabilities of gpt-5 on multimodal medical reasoning. *arXiv preprint arXiv:2508.08224*, 2025.

Michael Wornow, Rahul Thapa, Ethan Steinberg, Jason A Fries, and Nigam H Shah. EHRSHOT: An EHR benchmark for few-shot evaluation of foundation models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Chaoyi Wu, Pengcheng Qiu, Jinxin Liu, Hongfei Gu, Na Li, Ya Zhang, Yanfeng Wang, and Weidi Xie. Towards evaluating and building versatile large language models for medicine. *npj Digital Medicine*, 8(1):58, 2025.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus N Rabe, Charles Staats, Christian Szegedy, et al. Autoformalization with large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 32353–32368, 2022.

Ran Xu, Yuchen Zhuang, Yishan Zhong, Yue Yu, Xiangru Tang, Hang Wu, May Dongmei Wang, Peifeng Ruan, Donghan Yang, Tao Wang, et al. Medagentgym: Training llm agents for code-based medical reasoning at scale. In *The Second Workshop on GenAI for Health: Potential, Trust, and Policy Compliance*, 2025.

Xiao Yang, Xuejiao Zhao, and Zhiqi Shen. EHRStruct: A comprehensive benchmark framework for evaluating large language models on structured electronic health record tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 34340–34348, 2026. doi: 10.1609/aaai.v40i40.40731.

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 *International Conference on Learning Representations*, 2023.

A LIMITATIONS

Several limitations bound the current work. First, the benchmark is grounded in MIMIC-IV, a dataset from a single US academic medical center, and the autoformalized functions inherit the clinical conventions of that institution. Generalization to EHR systems with different schema designs, coding practices, or patient populations is not guaranteed without re-running the autoformalization loop. Second, the verification set used during autoformalization is small (approximately 100 examples per concept), which may allow rare edge cases to escape detection, although we intentionally design this split to mirror the scarcity and expensive nature of real medical data. Third, the natural-language guidelines used as input are generated by an LLM from PubMed abstracts. Although we manually inspected the summaries for quality, this adds a layer of potential drift from ground-truth clinical documents.

B BROADER IMPACTS

This work has potential benefits for clinical informatics by reducing the expert effort required to deploy LLM-based EHR analysis tools and by promoting reproducible, verifiable code over zeroshot generation. A library of verified clinical functions is more trustworthy than a sequence of one-off model outputs, which is relevant for safety-critical applications. At the same time, automated extraction of clinical concepts from EHR data carries risks if deployed without appropriate clinical validation: errors in autoformalized functions could propagate silently to clinical decision support systems. Additionally, MIMIC-IV is demographically skewed toward patients treated at Beth Israel Deaconess Medical Center in Boston, and function libraries trained on this distribution may perform less reliably for underrepresented patient groups. We recommend that any downstream deployment of autoformalized libraries be subject to institution-specific validation before clinical use.

C COMPARISON WITH EXISTING BENCHMARKS

Benchmark	Code Gen.	Scale (>10K)	Agentic	Compositional	Longitudinal
EHRSQL (Lee et al., 2022)	✓	✓	✗	✗	✗
EHRSHOT (Wornow et al., 2023)	✗	✓	✗	✗	✗
EHRAgent (Shi et al., 2024)	✓	✗	✓	✗	✗
AgentClinic (Schmidgall et al., 2024)	✗	✗	✓	✗	✗
MedAgentGym (Xu et al., 2025)	✓	✓	✓	✗	✗
MedAgentBench (Jiang et al., 2025)	✗	✗	✓	✗	✗
EHRStruct (Yang et al., 2026)	✓	✗	✗	✗	✗
AgentEHR (Liao et al., 2026)	✗	✓	✓	✗	✗
CodeClinic (Ours)	✓	✓	✓	✓	✓

Table 5: Comparison of EHR benchmarks relevant to clinical reasoning and agentic evaluation. **Code Gen.**: requires generating executable code or SQL. **Agentic**: supports iterative agent loop with execution feedback. **Compositional**: explicitly tests multi-hop reasoning over dependent clinical concepts. **Longitudinal**: evaluates reasoning over patient state as it evolves over time.

D BENCHMARK STATISTICS

Table 6: Compositional benchmark statistics.

	Level 1	Level 2	Level 3+	Total
Concepts	34	17	12	63

Question type	Count	Example question
Comparative	16,800	By how many ng/mL does this patient's maximum troponin T level exceed the myocardial injury threshold of 0.1 ng/mL?
Aggregation	15,600	What was this patient's maximum milrinone infusion rate (mcg/kg/min) during their ICU stay?
Temporal	14,400	How many hours after ICU admission was this patient first started on CRRT?
Direct	6,000	What is this patient's GCS motor component score on their first ICU day?
Count	4,200	How many distinct episodes of invasive mechanical ventilation did this patient have during their ICU stay?
Derived	3,400	What is the interval in hours between this patient's first antibiotic and first culture?
Ratio	1,800	What is this patient's BUN-to-creatinine ratio on their first ICU day?
Arithmetic	800	In which decade of life is this patient?

Table 7: Longitudinal ICU surveillance benchmark statistics. Counts are over the 26,000 checkpoints in the released 2,000-stay package unless otherwise noted.

Cohort		*Label distribution*		*Active clinical families*	
ICU trajectories	2,000	*Global action*		Infection	19,984
Checkpoints	26,000	Continue monitoring	5,886	Renal	16,568
Checkpoint interval	4 h	Escalate	20,114	Sepsis	14,585
Horizon per stay	48 h (13 ckpt.)	*Priority*		Respiratory	9,874
Median ICU LOS	93.6 h	Low	5,408	Hemodynamic	5,765
		Medium	10,585	Coagulation	5,505
		High	10,007	Metabolic	4,696
				Neurologic	1,389

Table 8: Complete specification of the 25 ICU surveillance findings across eight clinical families. Each finding has a deterministic builder over the community-validated MIMIC-Code derived schema that, at checkpoint time t, consumes only data with timestamp $\leq t$. **Class:** PE = persistent episode, CMS = cumulative max stage, AI = active interval, TTL = recent measurement with time-to-live, CCS = composite current state (recomputed each checkpoint). **Lvl:** S = suspect, A = alert. **Prov.** (provenance of the concept and threshold): *MC* = MIMIC-Code concept; *KDIGO*, *S3* (Sepsis-3), *Berlin* (ARDS), *SOFA*, *APACHE* (APACHE-III/APS-III), *SSC* (Surviving Sepsis Campaign) = published guideline or score whose cutoff we apply directly to the raw values; *INST* = our collaborating institution's clinical definition; *BM* = benchmark design. Concept values and clinical thresholds follow MIMIC-Code and the published scores it implements; the *temporal-semantics classes, TTL window lengths, composite shock states, and family/priority mapping* are our benchmark design.

Family	Finding	Lvl	Class	Trigger condition / threshold	Source & prov.
Infection	`infection_suspected`	S	PE	first suspected-infection time $\leq t$	`suspicion_of_infection` · MC
	`infection_confirmed`	S	PE	first positive culture time $\leq t$	`suspicion_of_infection` · MC
Sepsis	`sepsis_alert`	A	PE	first Sepsis-3 onset $\leq t$	`sepsis3` · MC/S3
Renal	`aki_stage1`	S	CMS	max KDIGO stage up to $t \geq 1$	`kdigo_stages` · KDIGO
	`aki_stage2`	A	CMS	max KDIGO stage up to $t \geq 2$	`kdigo_stages` · KDIGO
	`aki_stage3`	A	CMS	max KDIGO stage up to $t \geq 3$	`kdigo_stages` · KDIGO
	`oliguria_6h`	S	TTL	latest UO: `uo_mlkghr_6hr` < 0.5 (win. 6 h)	`urine_output_rate` · KDIGO
	`severe_oliguria_or_anuria`	A	TTL	latest UO meets severe 12/24 h oliguria/anuria (win. 24 h)	`urine_output_rate` · KDIGO/BM
	`crrt_active`	A	AI	CRRT mode active at t	`crrt` · MC
Respiratory	`resp_support_hfnc_or_niv`	S	AI	`ventilation_status` $\in$ {HFNC, NIV} overlaps t	`ventilation` · MC
	`resp_support_invasive_vent`	A	AI	`ventilation_status` $\in$ {InvasiveVent, Trach} overlaps t	`ventilation` · MC
	`hypoxemia_pf_lt_200`	S	TTL	latest PF ratio < 200 (win. 12 h)	`bg` · SOFA/Berlin
	`hypoxemia_pf_lt_100`	A	TTL	latest PF ratio < 100 (win. 12 h)	`bg` · SOFA/Berlin
Hemodynamic	`vasoactive_support_any`	S	AI	any vasoactive/inotrope interval overlaps t	`vasoactive_agent` · MC
	`vasoactive_multi_agent`	A	AI	≥ 2 vasoactive agents active at t	`vasoactive_agent` · BM
	`septic_shock_alert`	A	CCS	`sepsis` $\wedge$ vasoactive $\wedge$ lactate ≥ 2 (win. 12 h)	`sepsis3`+`vaso`+`bg` · INST/S3
	`shock_hypoperfusion_alert`	A	CCS	`sepsis` $\wedge$ vasoactive $\wedge$ lactate ≥ 4 (win. 12 h)	`sepsis3`+`vaso`+`bg` · INST/SSC
Neurologic	`gcs_moderate_impairment`	S	TTL	latest GCS in 9–12 (win. 8 h)	`gcs` · SOFA/std
	`gcs_severe_impairment`	A	TTL	latest GCS ≤ 8 (win. 8 h)	`gcs` · SOFA/std
Metabolic	`hyperlactatemia_ge_2`	S	TTL	latest lactate ≥ 2 (win. 12 h)	`bg` · S3/SSC
	`severe_hyperlactatemia_ge_4`	A	TTL	latest lactate ≥ 4 (win. 12 h)	`bg` · SSC
	`acidemia_ph_lt_7_30`	S	TTL	latest pH < 7.30 (win. 12 h)	`bg` · APACHE
	`severe_acidemia_ph_le_7_20`	A	TTL	latest pH ≤ 7.20 (win. 12 h)	`bg` · APACHE
Coagulation	`coagulopathy_inr_ge_1_5`	S	TTL	latest INR ≥ 1.5 (win. 24 h)	`coagulation` · BM
	`coagulopathy_inr_ge_2`	A	TTL	latest INR ≥ 2.0 (win. 24 h)	`coagulation` · BM

E GUIDELINE GENERATION FROM MEDICAL LITERATURE

For each clinical concept in the dependency graph, we generate a structured natural-language specification by querying PubMed. Given a concept name (e.g., "acute kidney injury"), we search PubMed for relevant clinical practice guideline articles and systematic reviews, retrieve 3–5 top-ranked results, and extract their abstracts and available full text. An LLM then synthesizes the retrieved literature into a structured specification containing:

(i) the clinical definition and diagnostic criteria with explicit measurement thresholds (e.g., serum creatinine increase ≥ 0.3 mg/dL within 48 hours per KDIGO guidelines),
(ii) the required EHR measurements mapped to MIMIC-IV table names and item identifiers,
(iii) severity staging criteria where applicable, and
(iv) temporal windows for event detection.

This pipeline produces guideline specifications for N concepts spanning the measurement, scoring, and condition layers of the dependency graph.

F CLINICAL KNOWLEDGE GRAPH CONSTRUCTION

We construct a directed acyclic dependency graph $\mathcal{G} = (V, E)$ where an edge $(v_i, v_j) \in E$ indicates that concept v_j requires the output of concept v_i. The graph is built in three phases.

Phase 1: Infrastructure nodes. We parse SQL definitions from the MIMIC-Code repository, extracting references to upstream derived tables, raw MIMIC-IV tables, and `itemid` values via regex matching to produce infrastructure nodes (measurements, medications, severity scores, treatments) with exact computational dependencies.

Phase 2: Clinical conditions. We expand the graph by mining frequent ICD-10 diagnosis codes among MIMIC-IV ICU patients, filtering for conditions detectable from structured EHR data, and generating guideline specifications using the pipeline in Appendix E. Dependencies for each condition are inferred by keyword-matching the specification text against infrastructure node descriptions and entity labels.

Phase 3: UMLS expansion. We map existing nodes to UMLS Concept Unique Identifiers and traverse one-hop Metathesaurus relations, filtering candidates by semantic type and ranking by centrality (the number of seed nodes connecting to each candidate).

The resulting graph contains multiple quality tiers—from infrastructure nodes with exact SQL-parsed dependencies, through conditions with guideline-validated dependencies, to automatically discovered conditions with heuristic dependency assignments—enabling both precise retrieval for core clinical concepts and scalability evaluation across a broad condition library.

G AUTOFORMALIZATION ALGORITHM

Algorithm 1 Autoformalize

Require: verification dataset $\mathcal{Q}$, database $\mathcal{C}$, guidelines $\mathcal{G}$, LLM $\mathcal{M}$, threshold θ, max iterations T

$\mathcal{S} \leftarrow \textsc{CodeSession}(\mathcal{C}, \mathcal{G})$; $f^* \leftarrow \emptyset$; $\alpha^* \leftarrow 0$; mem $\leftarrow []$
$msgs \leftarrow [\textsc{SysPrompt}, \textsc{UserPrompt}(\mathcal{Q})]$
for $t = 1, \ldots, T$ **do**
  if context near limit **then**
    mem $\leftarrow \textsc{Compress}(\text{mem} \cup \textsc{Summarize}(msgs))$
    $msgs \leftarrow [\textsc{SysPrompt}, \textsc{MemResume}(\text{mem})]$
  end if
  $r \leftarrow \mathcal{M}(msgs)$; $msgs \mathrel{+}= [r]$
  $\mathcal{B} \leftarrow \textsc{ExtractCodeBlocks}(r)$; **if** $\mathcal{B} = \emptyset$ **continue**
  $msgs \mathrel{+}= [\textsc{Execute}(\mathcal{B}, \mathcal{S})]$
  if `FINAL_FUNCTION` $\notin \mathcal{S}$.namespace **then**
    continue
  end if
  $f \leftarrow \mathcal{S}$.namespace[`FINAL_FUNCTION`]
  $(\alpha, \mathcal{E}) \leftarrow \textsc{Eval}(f, \mathcal{Q}, \mathcal{C}, \mathcal{M})$
  if $\alpha > \alpha^*$ **then** $f^* \leftarrow f$; $\alpha^* \leftarrow \alpha$
  end if
  if $\alpha \geq \theta$ **then break**
  end if
  $msgs \mathrel{+}= [\textsc{Feedback}(\alpha, \theta, \mathcal{E})]$
end for
return f^*

Context Compression. Because the ReACT loop may run for up to $T_{\max}$ iterations (default 100) with substantial code and query outputs at each step, the conversation history can exceed the LLM's context window. We address this with a rolling memory mechanism: when the estimated token count exceeds 25,000 tokens, the accumulated conversation is summarized by a separate LLM call into a compact memory document retaining key schema facts, the latest function source code, and a record of test results and error fixes. The full conversation history is then replaced by a two-message prompt (system instructions + memory resume). This compression is lossy but preserves the information most critical for continued refinement.

H PROMPTS

We document four prompt templates used throughout the paper: the autoformalization agent prompts (system and user, used to construct the Python concept library) and the evaluation prompts (system and user, used to score the generated functions on held-out questions). We then document the two prompt templates used by the longitudinal ICU surveillance agent at each 4-hour checkpoint.

H.1 AUTOFORMALIZATION AGENT

Autoformalization System Prompt

```
You are an expert clinical informaticist and Python programmer.
You can execute Python code by placing it inside <code> ... </code> XML tags.
The code runs in a persistent session with these pre-loaded: - `query_db(sql)`
-- runs a SQL query on the DuckDB database, returns a pandas DataFrame - `pandas`
(pd), `numpy` (np), `duckdb`, `datetime` are pre-imported - `DB_PATH` -- path to
the DuckDB database
IMPORTANT DATABASE NOTES:
{Database Specific Notes}
YOUR TASK: Write a Python function that extracts critical information for the
clinical concept described in the guideline you are given. The function should
use `query_db()` to look up patient data from the database and return useful
clinical information.
Steps:
1. **Explore the database schema** -- list tables, inspect columns, look up
dictionary/lookup tables (d_labitems, d_items, d_icd_diagnoses, etc.). Run
sample queries to understand data formats.
2. **Write a Python function** -- implement the guideline's logic. The function
should accept patient identifiers (e.g. subject_id, hadm_id, stay_id) and return
information that captures the clinical concept. The output format is flexible
-- return whatever best represents the concept (e.g., a value, dict, DataFrame,
boolean, list, etc.).
3. **Test thoroughly** -- call your function with several sample patients and
verify the output looks reasonable. Debug and fix any issues. Do NOT assign
FINAL_FUNCTION until you are confident it works.
4. **Save it** -- once you are satisfied, write a **single, self-contained
code block** that includes ALL imports, helper functions, and the main function
definition, then assigns FINAL_FUNCTION at the end. This block must be runnable
on its own -- do NOT include test calls, print statements, or assertions in it.
Example:
import pandas as pd
def helper(x): ...
def compute_concept(subject_id, hadm_id): ...
return value
FINAL_FUNCTION = compute_concept
5. After you receive evaluation feedback, **revise** your function, test the
revision, then submit a new self-contained code block assigning FINAL_FUNCTION
again.
IMPORTANT RULES:
- Your function MUST use `query_db()` to access the database.
- Do NOT call `duckdb.connect()` directly -- it will cause connection errors.
- Your function should accept patient identifiers and return information that
usefully captures the clinical concept. Any output format is fine.
- The function must work for any valid patient, not just the test cases.
- Do NOT guess column names or item IDs -- always look them up first.
- The code block that assigns FINAL_FUNCTION must be entirely self-contained (all
needed imports, helpers, and the function itself) and must NOT contain any test
calls, print statements, or assertions.
Be methodical. Explore first, code second. You may include multiple <code>
blocks in a single response.
```

Figure 8: The system prompt provided to the LLM agent during the autoformalization react loop. The prompt can be customized to include database specific information, such as important column names, primary key, etc.

H.2 COMPOSITIONAL INFORMATION SEEKING EVALUATION

H.3 LONGITUDINAL ICU SURVEILLANCE AGENT

Autoformalization Task Prompt

```
YOUR TASK -- build a function for the clinical concept '{concept_name}':
You need to write a Python function that can help answer clinical questions about
'{concept_name}' for individual patients.
Here are example questions your function will need to support:
{sample_questions}
APPROACH: 1. **Search for guidelines** -- call 'search_guidelines()' to
see what clinical guidelines are available, then use relevant keywords (e.g.
'search_guidelines("{concept_name}")') to find the most relevant ones. Read
each relevant guideline with 'get_guideline(name)' -- these contain clinical
definitions, scoring criteria, and implementation details that are essential
for getting the logic right. 2. **Check for reusable functions** -- call
'search_functions()' to see if any previously implemented functions could
serve as helpers. Use 'get_function_info(name)' to inspect their signatures
and docstrings, and 'load_function(name)' to import ones you want to reuse.
3. **Explore the database schema** -- list tables, inspect columns, look up
dictionary/lookup tables (d_labitems, d_items, d_icd_diagnoses, etc.). Run
sample queries to understand data formats. 4. **Write a Python function**
-- implement the clinical logic based on what you learned from the guidelines
and the database schema. The function should accept patient identifiers (e.g.
subject_id, hadm_id, stay_id) and return information that captures the clinical
concept. The output format is flexible -- return whatever best represents
the concept (e.g., a value, dict, DataFrame, boolean, list, etc.). 5. **Test
thoroughly** -- call your function with several sample patients and verify the
output looks reasonable. Debug and fix any issues. Do NOT assign FINAL_FUNCTION
until you are confident it works. 6. **Save it** -- once satisfied, write a
single self-contained code block with ALL imports, helpers, function def, and
FINAL_FUNCTION assignment. No test calls or print statements in that final
block.
FINAL FUNCTION REQUIREMENTS: - It should return useful clinical information
that captures the concept. - The input should involve some patient identifier
(e.g. stay_id) - The output can be any format (scalar, dict, DataFrame, etc.) --
choose whatever best represents the clinical concept. - Use 'query_db(sql)' or
'query_fhir(resource_type, params)' to look up data. - Have a detailed docstring
which explains the expected input and output.
Start by searching for guidelines relevant to '{concept_name}'.
```

Figure 9: The task prompt provided to the LLM agent to initiate autoformalization of a clinical concept. Template variables {concept_name} and {sample_questions} are filled at runtime.

Compositional Information Seeking Evaluation System Prompt

```
You are a clinical expert evaluating patient data for clinical concepts.
Execute Python code inside <code> ... </code> tags. The session pre-loads: -
'query_db(sql)' -- SQL query → pandas DataFrame - The concept function described
in your task - 'pandas' (pd), 'numpy' (np), 'datetime' pre-imported
Database: DuckDB with schemas 'mimiciv_hosp', 'mimiciv_icu', 'mimiciv_derived'.
Always use fully-qualified table names (e.g. 'mimiciv_hosp.admissions'). Never
call 'duckdb.connect()' -- use 'query_db(sql)'. IDs: subject_id (patient) →
hadm_id (admission) → stay_id (ICU stay).
```

Figure 10: System prompt for the compositional information seeking evaluation. The agent uses the autoformalized concept function loaded into the session to answer clinical questions about individual patients.

Compositional Information Seeking User Prompt

```
Session tools: - 'query_db(sql)' -- duckdb SQL query → pandas DataFrame
- 'search_functions(keyword="")' -- list available concept functions by
name - 'load_function(name)' -- load a concept function into the session
- 'get_function_info(name)' -- view a function's signature and docstring -
'search_guidelines(keyword="")' / 'get_guideline(name)' -- clinical guidelines
Concept functions relevant to this task are available -- use 'search_functions()'
to discover them and 'load_function(name)' to load what you need. You
should always check first to see if there are any useful functions for you
to use. {func_info} Patient: subject_id={subject_id}, hadm_id={hadm_id},
stay_id={stay_id}
{query}
Use the available functions and/or 'query_db' to answer, then reply with:
{verdict_format} For numerical questions, include as much decimal precision
as possible.
```

Figure 11: User prompt for the evaluation agent. Template variables are filled at runtime: {func_info} provides signatures of relevant concept functions, {subject_id}/{hadm_id}/{stay_id} identify the target patient, {query} is the clinical task, and {verdict_format} specifies the answer format.

Longitudinal Surveillance System Prompt

You are a general ICU rolling surveillance agent operating in rolling checkpoint mode. This is a rolling monitoring task, not a forecasting task. At each checkpoint, visible data and function outputs only contain information available up to the current checkpoint.

We monitor the following disease families:

- infection and sepsis
- renal injury and urine-output failure, including CRRT when relevant
- respiratory support escalation and hypoxemia
- hemodynamic instability, vasoactive support, and shock
- neurologic deterioration
- metabolic failure, including lactate elevation and acidemia
- coagulation abnormality

You are given the current checkpoint step index and time, a 'rolling_history' summarizing previous checkpoints, an embedded surveillance guideline digest, and a catalog of patient-state functions.

At each turn, return exactly one of:

1. a **tool call** to one focused patient-state function, when the current state is not yet sufficiently supported by 'rolling_history' or current-checkpoint evidence;
2. a **final decision** JSON, when the current state is sufficiently supported.

Do not return a final decision merely because no patient evidence has been retrieved. "Continue monitoring" is a clinical decision and must also be evidence-supported.

Decision semantics:

- suspected_conditions -- clinically meaningful concern that should keep monitoring focused on that condition family.
- alerts -- higher-acuity or higher-confidence states that justify escalation now.
- global_action -- exactly one of continue_monitoring or escalate. If alerts is non-empty, global_action should be escalate.
- priority -- exactly one of low, medium, high.

Final decision JSON contract: {"global_action":"continue_monitoring|escalate",
"suspected_conditions":["..."],
"alerts":["..."],
"priority":"low|medium|high",
"rationale":"..."}

Embedded guideline digest (excerpt):

- *Infection / Sepsis:* treat infection as an episode-level state once credible evidence appears; escalate to sepsis when infection is present and organ dysfunction (Sepsis-3 evidence or shock physiology) is strong.
- *Renal:* interpret AKI by KDIGO stage and remember the worst stage so far; oliguria is a rolling-window short-horizon signal; CRRT indicates active support, not a permanent state.
- *Respiratory:* HFNC/NIV is support-escalation; invasive ventilation is alert-level. PF ratio, oxygenation, lactate, pH expire if not refreshed.
- *Hemodynamic:* any vasoactive support is a strong instability signal; shock-like states depend on combined evidence (sepsis, support, metabolic).
- *Neurologic:* GCS $\leq$ 8 is a severe deterioration signal; GCS reflects recent observed status.
- *Metabolic:* lactate $\geq$ 2 = stress, $\geq$ 4 = alert; pH $<$ 7.30 = acidemia, $\leq$ 7.20 = severe.
- *Coagulation:* INR $\geq$ 1.5 = meaningful, $\geq$ 2.0 = alert-level coagulopathy.

Available patient-state functions (callable as {"tool_name":"call_function",
"arguments":{"function_name":"...",
"arguments":{"stay_id":N}}}): get_suspicion_of_infection, compute_sofa_score, kdigo_stages, get_urine_output_rate, ventilation_info, get_vasoactive_agent_info, gcs, get_blood_gas_info, get_coagulation, get_crrt_info.

A run_python fallback is available for focused custom queries against the checkpoint-scoped DuckDB session when call_function does not return enough information.

Figure 12: System prompt for the longitudinal ICU surveillance agent. The agent is invoked at every 4-hour checkpoint and must commit to either a focused tool call or a final structured monitoring decision. The full label vocabulary and the per-family guideline digest are embedded so the model has consistent definitions across checkpoints.

Longitudinal Surveillance User Prompt

The user message is a single JSON object describing the current checkpoint and the agent's prior actions for this stay:

```
{
  "step_input": {
    "trajectory_id": "{trajectory_id}",
    "stay_id": {stay_id},
    "step_index": {step_index},
    "t_hour": {t_hour},
    "task_name": "general_icu_surveillance"
  },
  "tool_backend": "{tool_backend}",
  "available_tools": [{available_tools}],
  "already_called_tools": [{tools_called_this_checkpoint}],
  "tool_outputs_in_order": [{tool_outputs_so_far}],
  "repeated_calls": [{repeated_call_warnings}],
  "rolling_history": {
    "0": "...short summary at t=0...",
    "4": "...short summary at t=4...",
    ...
    "{t_hour - 4}": "...summary at the previous checkpoint..."
  }
}
```

Field semantics:

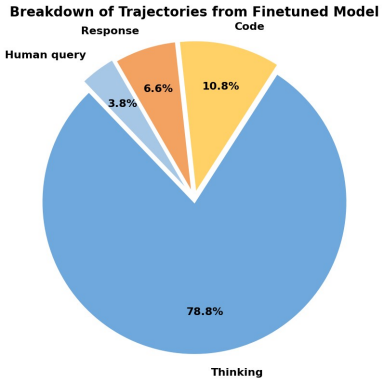
- t_hour -- the current checkpoint, in hours since ICU admission. The agent is run for $t \in \{0, 4, 8, \ldots, 48\}$.
- rolling_history -- a compact natural-language summary the agent itself produced at every *prior* checkpoint of this stay. null entries mean a family was not yet assessed and must not be treated as negative evidence.
- already_called_tools / tool_outputs_in_order -- the tool calls and tool outputs the agent has already issued *within the current checkpoint*, so it can chain evidence without re-querying.
- repeated_calls -- a nudge listing tool calls already issued at this checkpoint; the agent should not repeat them and should fall back to run_python if more evidence is needed.

The model must respond with exactly one tool call or one final decision JSON, conforming to the system prompt's contract. After every checkpoint the model also writes a one-sentence checkpoint_summary that becomes the next step's rolling_history entry.

Figure 13: User prompt for the longitudinal ICU surveillance agent at each 4-hour checkpoint. The payload is rendered as JSON; `rolling_history` accumulates the agent's own per-checkpoint summaries from earlier in the stay, so the model has direct access to its prior decisions when reasoning about the current state.



(a) Average rewards across levels during GRPO training. We see a consistent trend of improved performance with more training samples.



(b) Distribution of responses from finetuned model.

Figure 14: Findings from GRPO finetuning.

I Training Details

Category	Hyperparameter
Model	
Max prompt length	1024
Max response length	16384
Gradient checkpointing	Enabled
Model dtype	bfloat16
Training	
Algorithm	GRPO
Total epochs	1
Train batch size	4
PPO mini-batch size	4
Actor learning rate	1×10^{-5}
Clip ratio (low)	0.20
Clip ratio (high)	0.28
Clip ratio constant	10.0
Use KL in reward	False
KL coefficient	0.0
Use KL loss	False
KL loss coefficient	0.0
Dynamic batch size	Enabled
Rollout / Inference	
Backend	vLLM (async)
Tensor parallel size	2
Responses per prompt (train)	4
Responses per prompt (val)	1
Temperature (val)	1.0
Top-p (val)	0.6
Max user turns	5
Max assistant turns	5
Multi-turn format	Hermes

Table 9: Training and inference hyperparameters for the MIMIC-IV agent using GRPO.

Figure 14a shows the training progress using GRPO with our dataset. We believe more long-term training using our dataset would lead to strong gains in model performance. Figure 14b highlights a majority of token budget is spent on reasoning, followed by generating code.

J PRELIMINARY CROSS-DATABASE EVALUATION ON EICU

Our primary experiments use MIMIC-IV because its derived schema provides a broad set of reproducible clinical concepts with deterministic reference implementations. To examine whether clinical autoformalization is intrinsically tied to this database, we additionally conduct a preliminary cross-database evaluation on eICU.

We select two concepts, *Age* and *Blood Gas*, that can be defined in both datasets without relying on MIMIC-specific derived tables or institution-specific monitoring policies. We intentionally exclude concepts whose definitions depend on derived variables unavailable in eICU or on dataset-specific clinical logic, such as sepsis definitions requiring missing intermediate tables. The eICU schema and data organization differ substantially from MIMIC-IV, requiring the agent to rediscover the relevant tables and fields rather than directly reuse MIMIC-specific queries.

For this experiment, we use Qwen3.5-27B for both zero-shot inference and autoformalization. To limit additional computation, the autoformalization procedure is run for at most 16 refinement iterations per concept, substantially fewer than the budget used in our main experiments. No MIMIC-IV concept implementation is directly transferred to eICU; the agent is instead given access to the eICU database interface and constructs the corresponding function under the same general autoformalization procedure.

Table 10: Preliminary cross-database evaluation on eICU. Accuracy (%) is reported for zero-shot inference and clinical autoformalization using the same Qwen3.5-27B backbone.

Concept	Zero-shot	Autoformalization
Age	46.0	**65.0**
Blood Gas	36.2	**45.0**

Despite the change in schema and data organization, autoformalization improves accuracy on both evaluated concepts: by 19.0 percentage points for Age and 8.8 points for Blood Gas. These results suggest that the pipeline can be applied beyond MIMIC-IV with limited dataset-specific adaptation and does not rely solely on memorizing the organization of a single EHR database.

We emphasize that this experiment is intentionally limited in scope. Two concepts are insufficient to establish broad cross-institution generalization, and many higher-level clinical concepts cannot be transferred directly because their source variables, derived tables, or operational definitions differ across EHR systems. We therefore view this study as initial evidence of cross-database applicability rather than a comprehensive external validation. A broader evaluation across institutions, schemas, and clinical concept families is an important direction for future work.

K COMPUTE RESOURCES

All inference experiments were run on a `8xB200 Node`. Inference was served with `vLLM`. Approximate runtimes per experimental condition are as follows:

- **Autoformalization (per concept):** 20 GPU-hours on average for a `Qwen3.5-27B` model run with efficient parallelization via topological ordering of concepts.
- **GRPO Finetuning:** 144 GPU-hours total on a subset of the training split (50 questions per concept) for a total of 3150 questions. `4xH200` GPUs were used for training.
- **zeroshot inference evaluation (56,700 queries, one model):** 114 GPU-hours when using `Qwen3.5-27B` and parallel processing of 4 concepts at a time.

Total compute across all reported experiments (all models, all baselines) is estimated at $\sim$ 1000-2000 GPU-hours. No substantial additional compute was used for preliminary or failed experiments beyond the reported runs.

L SAMPLE INFERENCE TRAJECTORY

We present a representative inference trajectory drawn from the MIMIC-IV Demo dataset (publicly available, no credentialing required).[2] The trajectory uses `Qwen/Qwen3.5-27B-Instruct` as the inference model, consistent with the backbone used in the main experiments (Section 5). No pre-computed derived tables are available and no guideline text is pre-injected into the prompt — the agent must discover, retrieve, and apply clinical knowledge entirely through the guideline tools.

Table 11: Summary of the sample inference trajectory.

Field	**Value**
Concept	SOFA — CNS component (GCS-based)
Question	*"What is this patient's CNS (GCS-based) SOFA component score during their first 24 hours in the ICU?"*
Patient	subject_id=10007928, hadm_id=20338077, stay_id=35128235
Ground truth	**0** (GCS = 15, full consciousness)
Predicted	**0** ✓
LLM turns	7
SQL queries	2
Derived tables	None
Guideline pre-injected	No

[2]MIMIC-IV Demo is available at `https://physionet.org/content/mimic-iv-demo/` under the PhysioNet Open Data Commons License.

Trajectory Example: CNS SOFA Component Score (MIMIC-IV Demo)

Patient Query: *"What is this patient's CNS (GCS-based) SOFA component score during their first 24 hours in the ICU?"* **Label / Predicted:** 0 / 0 ✓

Agent Trajectory

[Search]: `search_guidelines("What is this patient's CNS (GCS-based) SOFA score during their first 24 hours in the ICU?")`

[Information]: `['first_day_sofa_guideline', 'saps-ii_guideline', 'coma_altered_consciousness_icu_guideline', ...]`
Semantic retrieval ranks `first_day_sofa_guideline` first over 351 section-level embeddings. Full clinical question used as query — not a short keyword.

[Read]: `get_guideline("first_day_sofa_guideline")`

[Information]: 500-char preview + section menu (not the full ∼6,000-token document):
- Definition - Diagnostic Criteria - **Scoring System**
- **Time Windows** - Severity Staging - Operational Notes

[Read Section]: `get_guideline_section("first_day_sofa", "Scoring System")`

[Information]: Exact GCS→score mapping retrieved (∼500 tokens):

GCS total	15	13–14	10–12	6–9	<6
SOFA CNS score	0	1	2	3	4

[Read Section]: `get_guideline_section("first_day_sofa", "Time Windows")`

[Information]: *"Use the **lowest** GCS documented within the 24-hour window (worst-value selection)."*

[Query DB]:
```
SELECT intime FROM icustays
WHERE stay_id = '35128235'  ⇒  2129-04-06 00:25:00
```

[Query DB]:
```
SELECT itemid, valuenum FROM chartevents
WHERE stay_id = '35128235'
AND itemid IN (220739, 223900, 223901)
AND charttime WITHIN first 24h
```

[Information]: Eye = 4, Verbal = 5, Motor = 6 at all 4 timepoints. Total GCS = 15 (consistent throughout window).

[Think]: From scoring table (Turn 3): GCS = 15 → SOFA CNS = 0. From time window rule (Turn 4): minimum GCS used — all readings are 15, so minimum = 15.

[Answer]: SOFA CNS score = **0**. *The agent retrieved only the two sections it needed (Scoring System and Time Windows) rather than loading the full guideline document.*

Figure 15: Sample inference trajectory on the MIMIC-IV Demo dataset. No guideline text is pre-injected; no derived tables are used. Model: `Qwen/Qwen3.5-27B-Instruct`.