CodeClinic: Evaluating Automation of Coding Skills for Clinical Reasoning Agents

Timothy Ossowski[1] **Xinchu Liu**[1] **Danyal Maqbool**[1] **Vaibhav Dhanuka**[1]
Sheng Zhang[2] **Hoifung Poon**[2] **Majid Afshar**[1] **Tyler Bradshaw**[1] **Junjie Hu**[1]
[1]University of Wisconsin–Madison [2]Microsoft Research

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. Compared with zero-shot code generation, the resulting libraries improve consistency while reducing per-query token usage by up to 40%.[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 showing strong capabilities across a diverse range of medical tasks Bedi et al. [2026], Wu et al. [2025], Wang et al. [2025]. Recent works have extended these capabilities to agentic settings, where LLMs interact with electronic health record (EHR) databases to answer clinically meaningful questions Liao et al. [2026], Tang et al. [2024]. These agents typically generate code or queries to extract structured information from EHR systems and reason over retrieved data to support clinical decisions.

A common approach is to equip agents with a user-designed toolbox of domain-specific functions for recurring clinical computations. While toolboxes improve reliability, they do not scale well: designing and maintaining them requires sustained expert effort and their coverage is inherently limited by the author's foresight. The alternative is a single-tool setup where the agent has only a generic interface, such as a single `query_db` function, without hospital-specific helper functions. However, relying only on this minimal single-tool setup can hurt both accuracy and cost, as the agent must repeatedly issue complex queries and re-derive intermediate results.

Beyond the toolbox size tradeoff, many clinically meaningful questions are not well-defined without additional context. Clinical concepts such as sepsis screening, organ failure scoring, and vasopressor

[1]Code to reproduce data and evaluation available at `https://github.com/tossowski/CodeClinic`

Preprint.

thresholds are not universal, with precise definitions varying across institutions and evolving with clinical guidelines over time. An agent answering such questions without access to the relevant institutional policies and practice recommendations faces an underspecified problem, and any benchmark grading its output against a single fixed ground truth will produce misleading accuracy estimates.

Thus, our benchmark aims to evaluate a central question: *Given limited verification data, how well can LLMs synthesize modular, composable libraries that operate over EHR data while dynamically accounting for institutional definitions and policies?* If successful, such models could remove the expert-curation bottleneck by generating code that adapts to location-specific clinical and health system policies.

We introduce CodeClinic, a benchmark built on MIMIC-IV Johnson et al. [2020] to evaluate this capability through two complementary tasks. The first is longitudinal Intensive Care Unit (ICU) surveillance, where an agent tracks patient state over time and issues structured monitoring decisions at 4-hour checkpoints across ICU stays, covering 25 findings across eight clinical families. The task simulates a doctor engaging in ICU patient monitoring, requiring persistent state tracking, handling time-sensitive variables, and recomputing composite conditions (e.g., sepsis) over a 13-step horizon without access to future information. The second is compositional information seeking, spanning 63 clinical concepts across nine domains and 63,000 queries, stratified by dependency depth to measure multi-step reasoning.

As a baseline, we propose an offline autoformalization pipeline that converts natural-language clinical guidelines into a reusable, verified Python function library via iterative LLM refinement. This library is constructed once using limited supervision and reused across all queries, ensuring consistency and reducing redundant computation.

Our contributions are as follows:

- **CodeClinic (Benchmark)**: a longitudinal agentic automation task for bedside ICU monitoring, paired with a compositional information seeking task which together surface the failure modes of LLM agents relying on static toolboxes and zeroshot code generation.
- **Benchmark Analysis (Findings)**: A systematic characterization of where and why current LLMs fail on compositional and longitudinal clinical reasoning.
- **Clinical Autoformalization (Baseline)**: An automated pipeline that transforms natural-language clinical guidelines into a verified Python function library using limited verification data, demonstrating a scalable alternative to user-designed toolboxes and serving as a strong baseline for future work on CodeClinic.

2 Related Work

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 Zhang et al. [2026]	✓	✗	✗	✗	✗
AgentEHR Liao et al. [2026]	✗	✓	✓	✗	✗
CodeClinic (Ours)	✓	✓	✓	✓	✓

Table 1: 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.

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 Bae et al. [2024]) and broader QA benchmarks (DrugEHRQA Bardhan et al. [2022], EHRXQA Bae et al. [2023], EHRNoteQA Kim et al. [2024]) cover constrained query spaces

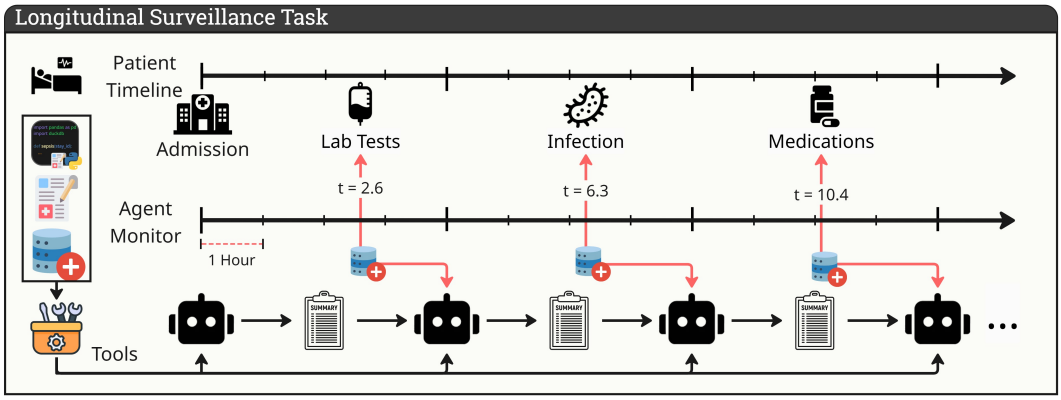


Figure 1: Illustration of the longitudinal ICU surveillance task in CodeClinic. The agent is prompted every 4 hours throughout an ICU stay to assess the patient's current condition, with strict visibility constraints that limit it to data available up to the current checkpoint. At each checkpoint, the 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.

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 Zhang 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. Table 1 summarizes the comparison; CodeClinic is the only benchmark to combine compositional dependency-depth stratification across 63 clinical concepts with longitudinal reasoning over full ICU trajectories.

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. Clinical definitions vary across institutions; although the natural-language guidelines we use draw on general consensus literature, the autoformalization loop grounds them in institution-specific practice by verifying generated code against labeled cases drawn from the target EHR system, fixing the ambiguity that makes zeroshot evaluation unreliable. 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.

3 Benchmark

CodeClinic is built around two complementary tasks over MIMIC-IV Johnson et al. [2023]: a *longitudinal ICU surveillance* task (Figure 1) that stresses agentic state tracking across rolling checkpoints, and a *compositional information seeking* task (Figure 3) that stresses multi-hop reasoning over the dependency graph of clinical concepts.

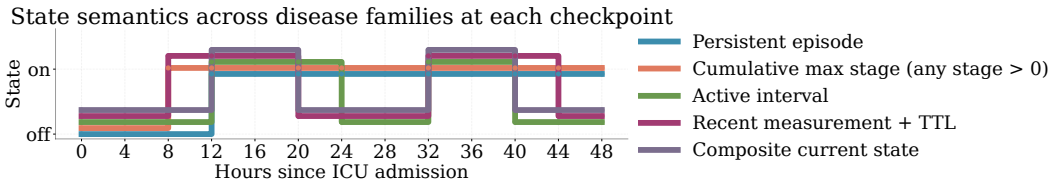


Figure 2: Example state dynamics across an ICU trajectory for different medical concepts. CodeClinic challenges models to interpret these changes in state dynamics across the patient timeline.

3.1 Longitudinal ICU Surveillance

The longitudinal task evaluates whether an agent can maintain clinically accurate state over time instead of answering a single isolated question. 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 compact interface is intentionally agent-friendly: it avoids committing to a fixed disease ontology while requiring the model to reason over the full multi-organ state at every step, rather than focusing on a single salient issue. The task also exhibits broad coverage. 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. The decision space spans both common and rare ICU conditions. The released 2,000-stay benchmark preserves realistic co-occurrence across families while ensuring sufficient support for clinically important but less frequent conditions such as Acute Kidney Injury (AKI), septic shock, severe acidemia, and severe coagulopathy.

Label Generation Internally, we maintain a registry of 25 canonical ICU surveillance findings, organized into eight clinical families that together cover the dominant ICU monitoring concerns: infection, sepsis, renal injury, respiratory support, hemodynamic support, neurologic deterioration, metabolic derangement, and coagulopathy. For each finding we define a deterministic builder over MIMIC-IV that consumes the patient's data up to the current checkpoint and decides whether the finding is currently active. The builder for each family encodes the appropriate clinical *temporal semantics* (Figure 2). Monotonic syndromes such as infection or sepsis remain active once they have been triggered within the trajectory. On the other hand, non-monotonic concepts such as ventilation or vasoactive support are active only while the intervention is in progress. A model therefore cannot solve the task by remembering a single binary label per disease. It must instead track which families persist once triggered, which families decay when evidence becomes stale, and which families are recomputed from multiple evolving components.

Agent Loop At each of the 13 checkpoints the agent sees the current hour, a compact summary of its own decisions and observations from previous checkpoints, and access to a toolbox for grounding its predictions in the database. The benchmark supports several agent-facing configurations: an agent can be asked to decide each checkpoint independently or with rolling history. Tools can expose curated MIMIC-IV concept queries, the autoformalized function library produced by our pipeline (Appendix F), or only the raw EHR tables, allowing us to separately measure the contribution of longitudinal context and of higher-level clinical tooling. In all cases the agent must retrieve the evidence it deems relevant, commit to a structured decision for the current checkpoint, and write a short summary that will be visible at the next step.

3.2 MIMIC-IV Compositional Information Seeking

We complement the longitudinal task with a clinical question answering benchmark grounded in the MIMIC-IV database. All questions are derived from tables in the MIMIC-IV `derived` schema,

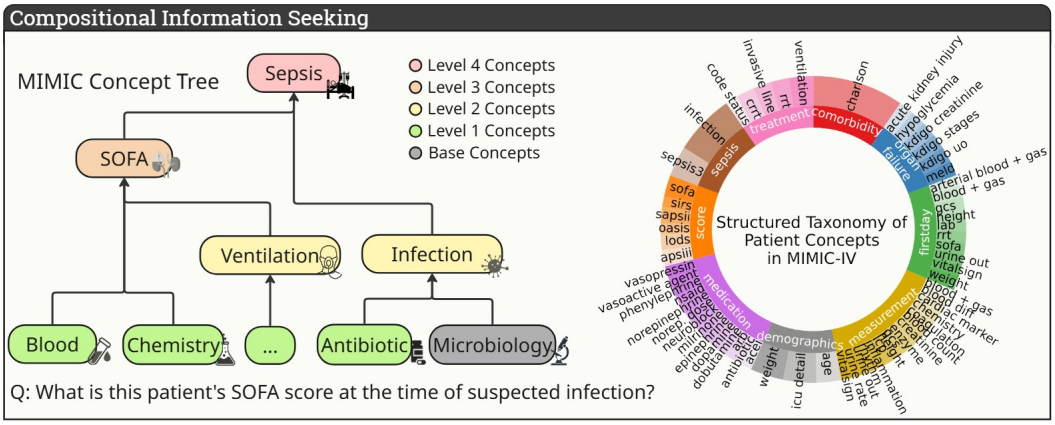


Figure 3: **Left:** Example sepsis-related question with its corresponding dependency graph. Answering the query requires composing intermediate clinical concepts (e.g., Sequential Organ Failure Assessment (SOFA) score used in organ dysfunction definitions for sepsis, suspected infection) that must be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

a curated collection of structured representations of the raw ICU data produced by community-validated SQL transformations applied to the original MIMIC-IV tables Johnson et al. [2021]. The benchmark spans 63 clinical concepts covering nine physiological and pharmacological domains: comorbidity scoring, patient demographics, first-day assessments, laboratory measurements, medication administration, organ failure indices, severity scores, sepsis-related variables, and ICU treatments.

Concept Definition Each concept corresponds to one derived table from MIMIC-IV (e.g., `sofa`) and is defined as a set of question variants paired with ground-truth extraction functions. Each concept specifies (i) a base SQL query that joins the derived table with `mimiciv_icu.icustays` and `mimiciv_hosp.admissions` to produce a per-stay feature matrix, and (ii) a set of question variants, each targeting a clinical attribute and question type. For each concept we also generate a natural-language guideline specification (defining clinical criteria, measurements, and time windows) by synthesizing PubMed literature with an LLM (details in Appendix D).

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. Depth-2 concepts (e.g., `ventilation`, `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. This hierarchy serves as a proxy for reasoning difficulty, with higher levels requiring increasingly complex, multi-step clinical inference. We provide a visual representation of these dependencies in Figure 3.

Clinical Knowledge Graph To determine dependency level difficulty, concepts are organized into a directed acyclic dependency graph $\mathcal{G} = (V, E)$, where an edge (v_i, v_j) indicates that v_j requires output from v_i. The graph is constructed by parsing SQL definitions in the MIMIC-Code repository and expanding with clinical conditions derived from ICD-10 diagnoses and UMLS Metathesaurus relations (see Appendix E for full construction details). This structure drives both difficulty stratification and compositional function reuse in our baseline pipeline.

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.

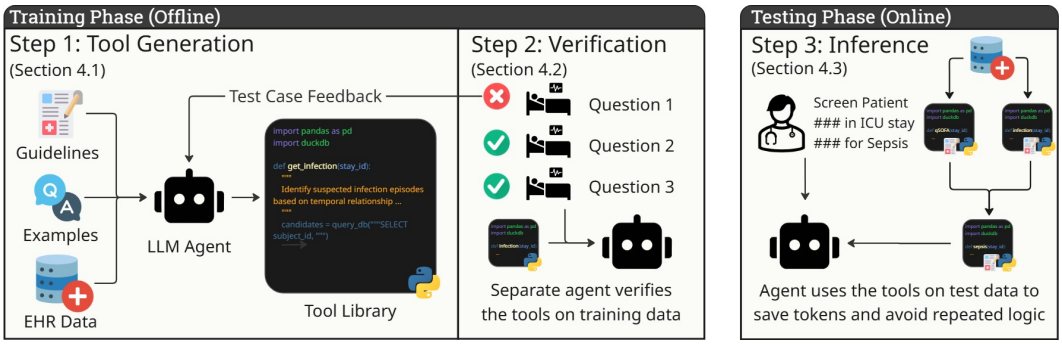


Figure 4: Overview of the autoformalization pipeline. During the offline *training phase*, an LLM agent uses medical guidelines, verification examples, and EHR data to construct a library of executable tools. A separate verification agent evaluates this library on the remaining training data, with errors 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.

4 Baseline: Clinical Autoformalization

We propose a clinical autoformalization pipeline (Figure 4) that converts natural-language clinical guidelines into a verified, 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 clinical guidelines $\mathcal{G}$, and produces a verified Python function f that deterministically extracts a clinical concept for any patient. The full algorithm is given in Appendix F.

4.1 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 clinical guidelines, 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. Guidelines are exposed as a searchable library rather than injected into the prompt, forcing the agent to bridge clinical knowledge and database schema through autonomous exploration.

4.2 Verification

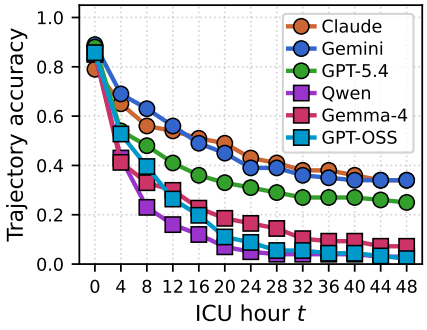
Once the agent produces a candidate function f_t, it is evaluated against the 10% train split of $\mathcal{Q}$. A separate agent calls f_t in its own code interpreter and parses a verdict for each question to compare against ground truth. If accuracy exceeds $\theta = 0.90$, the function is accepted 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, so the pipeline is robust to minor regressions between steps.

4.3 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 selects and calls the functions it deems relevant, reasoning over their outputs to produce an answer:

$$\hat{y} = \text{LLM}\big(q,\ \mathcal{L},\ \mathcal{C}\big) \tag{1}$$

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.



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	36.5	9.0	34.5	**26.4**	18.5
Qwen3.5-9B	33.0	7.7	31.4	23.8	20.4
Qwen3.5-27B	38.7	6.0	37.8	24.3	19.4
Gemma-4-31B	30.7	7.0	28.4	24.3	18.2
GPT-OSS-120B	**64.5**	**12.0**	**47.9**	24.7	**21.6**

Figure 5: **Left:** Trajectory accuracy on the rolling ICU surveillance task declines as monitoring windows lengthen. **Right:** Summary of longitudinal monitoring metrics (zeroshot method).

5 Results

For our experiments we consider the following baselines. Each baseline is allowed up to 15 conversation turns of exploration (10 for longitudinal task) before being prompted to give a final answer. We provide an example trajectory in our environment in Appendix J:

(1) Zeroshot: The LLM receives only the raw MIMIC-IV schema and a `query_db` function.
(2) Clinical Autoformalization (Ours): The LLM is given the raw MIMIC-IV schema, a `query_db` function, and the autoformalized function library $\mathcal{L}$ generated from the compositional information seeking train split. We use the same library created with `Qwen3.5-27B` across all runs.

Evaluation Metrics: For the **longitudinal task**, each checkpoint is scored on global-action accuracy (**Acc**), priority level accuracy (**Prio**), and macro set-based F_1 over the predicted suspected-conditions and alerts sets (**Susp. F1**, **Alerts F1**). *Trajectory accuracy* (**Traj. Acc**) is a stricter metric measuring the fraction of stays where the global action is correct at *every* checkpoint, so a single error anywhere in the 13-step horizon counts as a full 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. Note, we found that the GRPO finetuned model struggled to generalize to the new task, so we limit our analysis to zeroshot and autoformalization methods. We observed the following findings:

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. Our design forces the agent to keep monotonic syndromes (infection, sepsis) active once triggered while letting other states (oliguria, hyperlactatemia, severe acidemia) decay. Models that track only the latest checkpoint or only the running maximum are penalized by both regimes simultaneously.

Disease tracking is harder than predicting actions: Across all models, global-action accuracy ($\sim$ 25–76%) is much higher than suspected-conditions F_1 ($\sim$ 18–43%) and alerts F_1 ($\sim$ 22–47%). These metrics probe whether the model can produce the *full multi-organ picture* at each step rather than only the dominant problem, which is the clinically actionable signal in real ICU monitoring.

Autoformalized functions improve longitudinal consistency. As shown in Figure 6, the majority of models equipped with the autoformalized library achieve higher step accuracy than their zero-shot counterparts, with gains in token efficiency across all models. Because the pre-verified functions encode the correct temporal semantics of each clinical concept, the agent no longer needs to re-derive these rules from scratch at each checkpoint, reducing the per-step error rate.

[1]We ensured that no data was saved or used for training to comply with the MIMIC-IV data usage agreement. Frontier models also were run on a 21% stratified subset of the data due to prohibitively expensive cost.

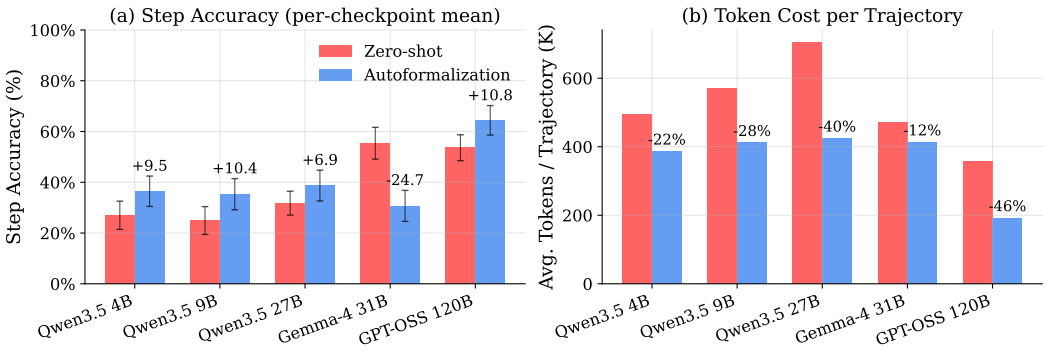


Figure 6: Step-level action prediction accuracy of the zeroshot versus autoformalization baseline comparison on the monitoring task. Error bars: 95% Wilson score interval ($z = 1/96$).

5.2 Compositional Information Seeking Analysis

To characterize the difficulty of CodeClinic's compositional information seeking component, we evaluate a diverse set of models in a zero-shot code-generation setting. Each model is given the raw MIMIC-IV schema, a `query_db` function, and a clinical question, and must iteratively generate executable code to retrieve and analyze information from the database. The model can then reason over execution outputs to produce an answer or continue refining its code, for up to 15 iterations. We select models spanning multiple families and scales to ensure broad coverage and to control for potential schema memorization from pretraining data, since MIMIC-IV is a widely studied dataset.

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: Zeroshot accuracy (%) of LLMs on the CodeClinic compositional information seeking task by difficulty level. All models are given only the raw MIMIC-IV schema and a `query_db` tool.

Table 2 reports accuracy across 63 concepts by difficulty level. Several trends emerge. First, even the strongest frontier models achieve only moderate accuracy, confirming that clinical information seeking over real EHR databases remains challenging. The best-performing model (Claude-Sonnet-4.6) reaches 53.1% overall accuracy, well below expert-level performance, indicating substantial headroom for future systems. Second, performance consistently declines from Level 1 to Level 3+ across all model families, with the drop (column Δ) reflecting the compounding effect of multi-hop dependencies. For example, answering a Sepsis-3 question requires correctly computing SOFA sub-scores, infection criteria, and antibiotic timing, each of which can fail independently.

Comparison with Finetuning Table 3 compares a traditional RL finetuning approach against our autoformalization pipeline, alongside a zeroshot baseline, to isolate the effect of learning reusable code libraries versus adapting the model weights directly. All approaches use `Qwen3-4B` as the backbone LLM. For GRPO finetuning, we finetune `Qwen3.5-4B` Yang et al. [2025] using standard GRPO Shao et al. [2024] with 50 training examples per concept and rewards based on exact match

Method	Accuracy by Level			Overall Acc.	Tokens/Q	Time/Q (s)
	Level 1	Level 2	Level 3+			
Zeroshot	44.9	32.8	26.3	38.1	30479	3.1
GRPO Finetuning	**57.1**	36.5	27.9	40.5	**22376**	2.7
Autoformalization (Ours)	51.2	**41.5**	**34.1**	**45.3**	23569	**2.5**

Table 3: Comparison of approaches across all 63 concepts on the held-out compositional information seeking test set. Accuracy is macro-averaged across concepts. Token usage and time are reported per question at inference.

between the predicted and ground truth answer. Finetuning improves Level 1 accuracy, likely by teaching shallow database search patterns, but performance drops substantially on Level 2+ tasks requiring deeper compositional reasoning. In contrast, our autoformalization pipeline achieves performance competitive with finetuning without parameter updates, demonstrating that the offline verification loop can synthesize useful reusable functions. Autoformalization also reduces per-query token usage by approximately 23% compared to zeroshot inference, since the agent invokes compact verified functions instead of repeatedly rediscovering schema logic. Full training details are provided in Appendix H.

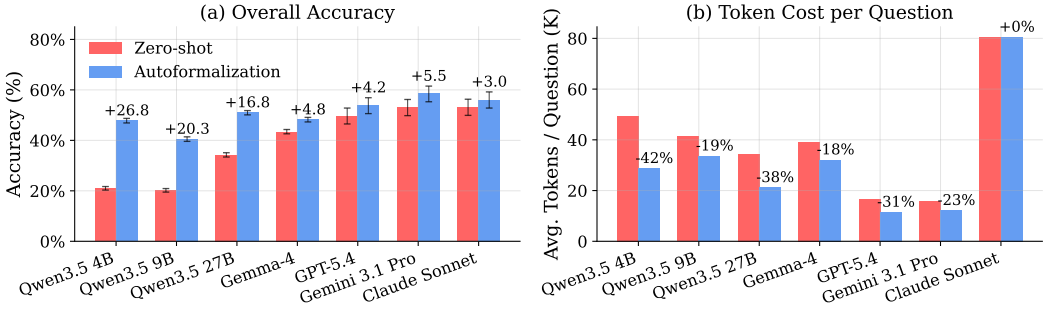


Figure 7: Performance comparison between zeroshot evaluation and our autoformalization baseline on the compositional information seeking task. Autoformalization shows universal performance improvement while improving or matching token efficiency across all models. Error bars: 95% bootstrapped interval across 2000 resamples.

Autoformalization Details As observed in the longitudinal task, Figure 7 illustrates that accuracy gains from autoformalization in information seeking are consistent across model families, with the relative benefit most pronounced for smaller models which frequently fail to chain dependent clinical computations correctly. Token efficiency similarly improves or matches zeroshot across models, confirming that the offline verification cost is amortized over many queries and does not inflate per-question inference cost. The cross-model consistency of these gains suggests that the benefit derives from the verified functions themselves and motivates autoformalization as a general-purpose approach for grounding LLM agents in clinical code libraries.

Conclusion

We presented CodeClinic, an agentic benchmark consisting of 65k coding instances spanning 260 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.

References

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.

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*, pages 1–9, 2026.

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.

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.

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.

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*, pages 599–621, 2024.

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.

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.

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.

Wenxing Shi, Yinan Xu, Yuchen Zhuang, Yue Yu, Junfeng Zhang, Hui Wu, Joyce C Ho, Carl Yang, and Chao Zhang. EHRAgent: Code empowers large language models for complex tabular reasoning on electronic health records. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 6315–6325, 2024.

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.

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.

Yixing Jiang, Kameron Bhatt, Brendan Rousseau, Andrew Phelps, Jonathan H Shen, Nigam H Shah, Jonathan H Chen, Jure Leskovec, Jason A Fries, Alison Callahan, and Leon A Gatys. MedAgentBench: A realistic virtual EHR environment to benchmark medical LLM agents. *NEJM AI*, 2025. arXiv:2501.14654.

Xinlu Zhang, Zhiyu Li, Yifan Yang, Siyuan Wang, Jiaru Chen, and Xinya Li. EHRStruct: A comprehensive benchmark for evaluating large language models on structured electronic health record tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. arXiv:2511.08206.

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.

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, page 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.

Jaehee Bae, Gyubok Lee, and Edward Choi. EHR-SeqSQL: A sequential text-to-SQL dataset for electronic health records. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

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, editors, *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 1083–1097, Marseille, France, June 2022. European Language Resources Association. URL https://aclanthology.org/2022.lrec-1.117/.

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.

Ji-Youn Kim, Sung-Min Shim, Jiyeon Park, Yeram Hwang, Min-Sung Kim, Yun-Geun Kim, and Jong Chul Ye. EHRNoteQA: A patient-level question answering benchmark for real-world clinical practice using discharge summaries. In *Advances in Neural Information Processing Systems*, 2024.

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.

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.

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*, pages 2025–05, 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, pages 32353–32368, 2022.

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.

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*, pages 10764–10799. PMLR, 2023.

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.

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.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL `https://arxiv.org/abs/2505.09388`.

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.

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 Benchmark Statistics

Table 4: 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?

	Trajectories	Checkpoints	Interval	Horizon	Median ICU LOS
Cohort	2,000	26,000	4 h	48 h (13 ckpts)	93.6 h

	Continue monitoring	Escalate
Action	5,886	20,114

	Low	Medium	High
Priority	5,408	10,585	10,007

Table 5: 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

D 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.

E 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 D. 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.

F 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.

G 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.

G.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.

G.2 Compositional Information Seeking Evaluation

G.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..."
  }
}
```

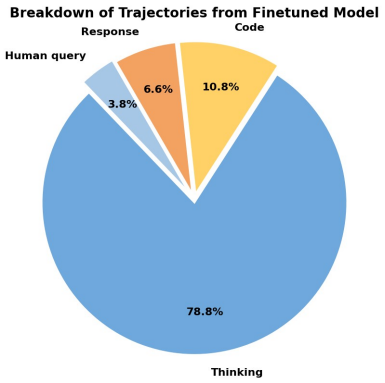
Field semantics:

- `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.

H 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 6: 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.

I 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.

J 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 precomputed 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 7: 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`.