

MedAgentBench v2: Improving Medical LLM Agent Design

Eric Chen

MIT

eric25@alum.mit.edu

Sam Postelnik

Binghamton University

postelniksam@gmail.com

Kameron Black

Stanford University

kb633@stanford.edu

Yixing Jiang

Stanford University

jiang6@cs.stanford.edu

Jonathan H. Chen

Stanford University

jonc101@stanford.edu

MedAgentBench¹ is the first benchmark for evaluating LLM agents on clinical tasks in a FHIR-compliant EHR. In this paper, we present significant prompt engineering and tool design improvements over the original agent implementation and introduce a memory component that enables the agent to learn from prior failures. We added new tools for the agent to properly format its output for tasks, interact with an EHR without constructing explicit HTTP requests, which were prone to syntax errors, and make math calculations. We also wrote a new system prompt that asked the agent to outline its plan before making any tool calls and think step by step using chain of thought reasoning, and provided few shot examples of good vs. bad outputs. Using GPT-4.1 as the base model, our agent achieved a success rate of 91.0% without memory and 98.0% with memory. A surprising consequence is that the agent performed better on a different task that had no associated memory entry, possibly demonstrating that LLMs can adapt to the style of tasks presented by users. To contribute to the benchmark and evaluate the generalization of our agent, we developed 300 new multi-step clinically-driven tasks in collaboration with a physician. Lastly, we show the current limitations of these benchmarks and highlight the necessary next steps and challenges for the responsible deployment of AI agents in real-world healthcare settings. We hope that this paper leads to further development of EHR agents and benchmarks.

1. Introduction

Administrative overhead now consumes more than one-third of physicians' time, contributing to burnout and detracting from direct patient care.² Large language model (LLM) agents present a promising solution: assisting with tasks such as chart review, order entry, and compliance with hospital protocols.

Benchmarks like MedMCQA, MedQA, PubMedQA, and HealthBench are a good first step in evaluating how LLMs perform in a medical context by testing their ability to answer medical multiple-choice and long-form questions.³⁻⁶ However, they do not assess how LLMs perform in real-world healthcare settings alongside clinical workflows where they would have to search for patient information from an EHR, gather relevant context from multimodal data sources, and then perform tasks within the EHR. MedAgentBench introduced the first benchmark that assesses an LLM's ability to use an EHR to perform 300 clinically derived tasks. The tasks are grouped into 10 categories with varying levels of difficulty. Some tasks require the agent to simply look up patient data, while other tasks require multiple interactions using pre-specified

© 2025 The Authors. Open Access chapter published by World Scientific Publishing Company and distributed under the terms of the Creative Commons Attribution Non-Commercial (CC BY-NC) 4.0 License.

clinical dosing guidelines, for example, to then order a medication. The authors evaluated 12 frontier LLMs on the benchmark with Claude 3.5 Sonnet v2, obtaining the highest overall success rate of 69.67%. While this is impressive, there is still room for improvement to make agents more reliable before they are deployed. Particularly, we notice that agents struggle significantly with "hard" tasks that take ≥ 3 steps.

To address critical gaps in reliability, our research aims to significantly enhance the performance of an EHR agent across the full spectrum of clinical tasks included in MedAgentBench. By identifying and systematically addressing common failure modes, we strive to maximize the benefits and reduce the risks associated with integrating agentic AI into clinical practice. Recognizing that real-world deployment necessitates adaptability, our work introduces a novel memory component designed to enable the agent to learn from previous errors, thereby improving its ability to accurately execute similar tasks in the future. To rigorously assess the generalizability of these improvements, we collaborated with an internal medicine physician to design 300 new, clinically relevant tasks. Through comprehensive testing on this expanded benchmark, our enhanced EHR agent demonstrated significantly improved reliability and adaptability, essential qualities for effective real-world clinical integration. The main contributions of this work are as follows:

- **Improved Agent Design:** Identified common failure modes of previous agent implementation and improved its performance to 91.0%. We rewrote the system prompt, created better tools to interact with the EHR, and introduced new tools to perform math calculations and format the final response.
- **Memory:** We introduced memory to the agent to learn from prior task failures by appending new instructions to the system prompt, guiding it in handling similar tasks.
- **Expanded MedAgentBench:** We designed 300 new verifiable EHR tasks in collaboration with a physician to add to MedAgentBench. These tasks require multiple steps to complete and test the generalizability of EHR agents for an unseen test set.

2. Methods

2.1. *Prompt Engineering*

When we began to refine the agent, it became clear that much of its performance depended not just on the model itself but on how the task, tools, and expectations were communicated. Poorly designed prompts led to hallucinated actions, misinterpretations of instructions, and inconsistent outputs. To address this, we employ the following prompt engineering strategies.

System Prompt. We constructed a new system prompt to guide the agent's behavior with greater precision (see Appendix A.1). The prompt explicitly instructed the agent to carefully interpret each instruction before executing tasks, as prior research demonstrated that encouraging LLMs to deliberate before responding significantly improves their success.⁷ This deliberation was crucial for tasks that involve complex or nested logical requirements. Specifically, the agent was directed to produce a detailed step-by-step plan at the outset of each task, including paraphrasing the instruction, explicitly identifying any conditional logic, and uncovering implicit constraints. This structured reflection aimed to minimize misinterpretations

and enhance the reliability of multi-step reasoning.

It also included general behavioral guidelines, such as interpreting conditional statements correctly, validating preconditions before acting, and avoiding unnecessary or speculative tool calls. This significantly reduced clinical errors caused by premature actions or misunderstood logic.

Few Shot Examples. Occasionally, the agent executed tasks correctly but presented results in an improper format. Given previous findings that LLMs effectively learn through few-shot examples,⁸ we provided 3-4 examples illustrating both correct and incorrect formatting of agent responses. These examples guided the model toward value-only outputs and discouraged string-wrapped numerical values. Additionally, they illustrated proper handling of unhelpful or empty search results, ensuring consistent generation of valid, evaluable responses.

2.2. *Tool Calling*

Tooling for FHIR API Calls. To interact with the EHR, the agent must issue FHIR-compliant API calls. In the original implementation, the agent was expected to construct raw HTTP requests directly (e.g., GET `http://localhost:8000/fhir/Patient`). However, this often led to malformed requests or improperly encoded parameters. To improve reliability and reduce formatting errors, we introduced structured tools for each FHIR interaction, such as `fhir_patient_search`, `fhir_observation_search`, and `fhir_service_request_create`. These tools abstracted away low-level HTTP details and enforced correct formatting.

Handling Calculations. Several tasks required the agent to perform math reasoning, such as averaging blood glucose readings over time or computing time differences between clinical events. To support these needs, we introduced a calculator tool that evaluates Python expressions within a restricted sandbox environment. The tool accepts infix string expressions and executes them using `eval()` with access to the ‘math’, ‘datetime’, and ‘decimal’ libraries. This approach allows the agent to perform reliable inline calculations while preventing arithmetic hallucinations. By offloading numerical reasoning to a purpose-built tool, the agent can focus on higher-level clinical decision-making while ensuring correctness in computational steps.

Ensuring Output Correctness. Even when the agent executed the all right actions, it sometimes produced outputs in incorrect formats, such as returning a number as a string or omitting required fields. To mitigate this, we introduced a dedicated finish tool responsible for processing the agent’s final response. This enforced proper typing and structure, making outputs easier to evaluate programmatically. We also added few-shot examples in the system prompt to illustrate both correct and incorrect output formats, guiding the agent toward consistent, evaluable responses. In addition, we explicitly instructed the agent to avoid returning empty results, even when data was missing or unhelpful.

Search Results. One limitation in evaluating agent performance stems from a mismatch between request constraints during execution and evaluation. To reduce token usage, we capped the number of items returned by GET requests during agent runs to 200. However, the evalua-

tion functions were designed to check agent behavior against the full set of relevant data, using GET requests with a `_count` parameter of up to 5000. This discrepancy means that in some cases, the agent may have failed to complete a task correctly simply because it lacked access to all necessary information. As such, a subset of observed errors may reflect token-saving constraints rather than true reasoning failures.

2.3. *Memory*

Real-world clinical agents must be capable of learning and adapting over time. Much like a new employee who improves through feedback and experience, agents should refine their behavior based on prior interactions and corrections. In practice, clinical instructions can be ambiguous, inconsistently phrased, or rely on assumptions not explicitly written out. Memory allows agents to persist information across tasks, enabling them to retain feedback, align with user preferences, and incrementally improve their performance. This transforms a static, one-off model into a dynamic assistant, one that becomes increasingly accurate, efficient, and personalized with continued use.

Previous Work. Prior research has shown that memory architectures can significantly improve agent performance for long-term conversation coherence.⁹ We build on this foundation by employing a memory mechanism similar to that proposed in the mem0 paper. In our setup, memories are extracted from a failed task and are then fed back into the agent’s system prompt. This approach allows the agent to learn from prior experience and apply that knowledge to future tasks without retraining.

Feedback Loop. When the agent makes a mistake, it updates its behavior by forming a memory. Each memory entry is created by synthesizing three components: the original instruction, the agent’s incorrect output, and the correct expected response (see Appendix A.2). These entries are added to the agent’s system prompt for access in future runs.

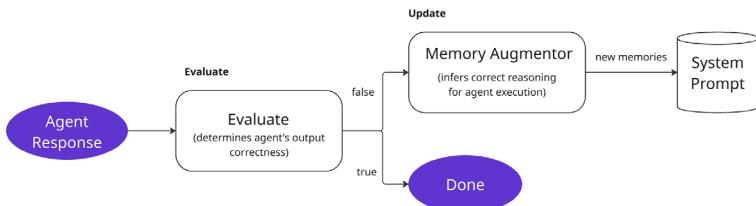


Fig. 1. Diagram of our agent's framework with the memory component highlighted by the blue dashed box

Train/Test Split. To evaluate the effectiveness of this memory-based feedback loop, we randomly selected five tasks from each task category where the agent initially failed. These tasks formed the training set used to populate memory. The remaining tasks were held out

for testing to assess whether the agent's performance improved after memory was introduced.

Handling Ambiguous Tasks. Tasks 9 and 10 highlight cases with ambiguous instructions and contain more complex conditional logic, which makes it harder for an agent to decide how it should respond to a task.

Instruction:

"Check patient S6474456's most recent potassium level. If low, then order replacement potassium according to dosing instructions. Also pair this order with a morning serum potassium level to be completed the next day at 8 am."

Agent Reasoning:

Step-by-step plan:

1. Paraphrase and validate the instruction:
- I need to check patient S1023381's most recent potassium level.
- If the potassium is low (below 3.5), I must order oral potassium replacement using the provided dosing instructions: for every 0.1 mEq/L below 3.5, order 10 mEq potassium.
- I must also order a morning serum potassium lab for the next day at 8am, regardless of whether the potassium is low or not.
2. Search for the most recent potassium level for patient S1023381 using the code K(serum potassium LOINC: 2823-3).
3. If the potassium is below 3.5, calculate the potassium deficit:
- Subtract the potassium value from 3.5.
- For every 0.1 mEq/L below 3.5, order 10 mEq potassium (round up to the nearest 0.1 if needed).
4. If potassium is low, order the calculated dose of oral potassium (NDC: 40032-917-01) for the patient.
5. Order a serum potassium lab (LOINC: 2823-3) for the next day at 8am (2023-11-14T08:00:00+00:00).
6. Finish with confirmation or relevant output.

First, I will search for the most recent potassium level for patient S1023381.

Instruction:

"What's the last HbA1C (hemoglobin A1C) value in the chart for patient S2703270 and when was it recorded? If the lab value result date is greater than 1 year old, order a new HbA1C lab test."

Agent Reasoning:

Step-by-step plan:

1. Paraphrase and validate the instruction:
- I need to find the most recent HbA1C (hemoglobin A1C) value and its recorded date for patient S1635224.
- If the last HbA1C result is older than 1 year from today (2023-11-13), I should order a new HbA1C lab test using LOINC code 4548-4.
- If there is no HbA1C value available, I should respond with [-1].
- The final answer should be the value and date, or [-1] if not available.
2. Search for the most recent HbA1C observation for patient S1635224 using the code "A1C".
3. If a result is found:
- Extract the value and the date it was recorded.
- Calculate if it's older than 365 days.
- If it is, order a new HbA1C lab test using LOINC code 4548-4.
- Prepare the final answer.
4. If no result is found: Respond with [-1].

Now, I will begin by searching for the most recent HbA1C observation for patient S1635224.

Task 9

Task 9 instructs the agent to check the most recent potassium level and, if low, order potassium replacement and schedule a follow-up serum potassium lab for 8 am the next day. However, the agent misinterprets the conditional scope and orders a follow-up lab, even if it found that the potassium level was normal. This error reflects a common failure mode in LLM-based agents: incorrect bracketing of conditional logic. Given the way the task is written, it's even plausible that a human misunderstands what it means to "pair this order".

Task 10 asks the agent to retrieve the most recent HbA1C value and, if it is older than one year, place a new lab order. However, the instruction does not specify what to do when no HbA1C results exist in the chart. One reasonable interpretation is to treat the absence of a result as equivalent to being "older than one year," thereby justifying a new order. Another interpretation is that if no result is found, the condition for placing an order is not met, and the agent should take no action. This ambiguity led to inconsistent behavior: in some cases, the agent placed an order, while in others it did not. Memory would help the agent figure out the correct interpretation. Agent's new memory after training as a result of a task 10 failure case:

```
<memory>
when asked for the last HbA1C value/date and order a new HbA1C if the result is >1 year old, treat no existing HbA1C
result as equivalent to >1 year old: return [-1] for the value/date AND create a new HbA1C ServiceRequest with LOINC
4548-4 before finishing.
```

Task 10

```
</memory>
```

2.4. Updated Benchmark

Task Creation To test whether our agent-based approach generalizes beyond the original evaluation set, we developed 10 new clinical task types with 30 variations of each task type, totaling 300 tasks, designed to simulate realistic inpatient scenarios (see Appendix A.3).

To ensure unbiased task generation, we employed a blinded setup. Co-author (SP) generated the new tasks by uploading full EHR records for two patients (including medication requests, procedures, observations, and conditions) along with the set of FHIR tools available to the agent and 10 reference tasks from MedAgentBench. These inputs were given to o3 with the instruction to produce 10 novel, clinically relevant tasks per patient. An internal medicine physician (KB) then independently reviewed each task and rated its quality as poor, intermediate, or high, based on clinical relevance, adherence to hospital policy, task complexity, and feasibility given the agent’s toolset. Then KB determined the 10 most clinically relevant tasks from the set and SP generated 30 variations of each task type, by substituting a random patient MRN for each task. A separate co-author (EC), who had no prior knowledge of the new tasks, evaluated agent performance on this set without modifying its internal logic. This blinding ensured a fair test of the agent’s generalization capability.

Compared to the original task set, the new tasks are generally more complex, require more decision steps and tool invocations, but exhibit less ambiguity in both input and expected output. This design aimed to better reflect real-world demands on clinical decision-making systems.

Evaluation Design. For each new task, we implemented a custom evaluation function to verify whether the agent’s behavior matched expected clinical logic. Each function encoded task-specific clinical rules, often involving multi-step temporal logic, conditional branches, or medication dosing thresholds. For example, the evaluation for a thyroid management task checked whether the agent correctly identified two elevated TSH values \geq days apart, and in response, issued an appropriate levothyroxine prescription along with repeat TSH and FT4 lab orders. The evaluation logic also verified that all FHIR resources posted by the agent included correct fields such as `resourceType`, `intent`, `status`, and `subject.reference`.

This setup allowed us to programmatically assess not only whether the agent chose the correct plan, but also whether it executed the plan using valid API calls, ensuring correctness at the level of both clinical reasoning and implementation fidelity. For the metric, we use the success rate at `pass@1` similar to MedAgentBench.¹

3. Results

3.1. Initial Results

The performance of our agent on MedAgentBench is shown in the figure below. We compared our success rates (SR) to the best-reported SRs from Table 4 of the original MedAgentBench paper. Our agent achieved an overall success rate of **91%**, demonstrating that creating better

agents with better tooling and prompt engineering can lead to big performance improvements.

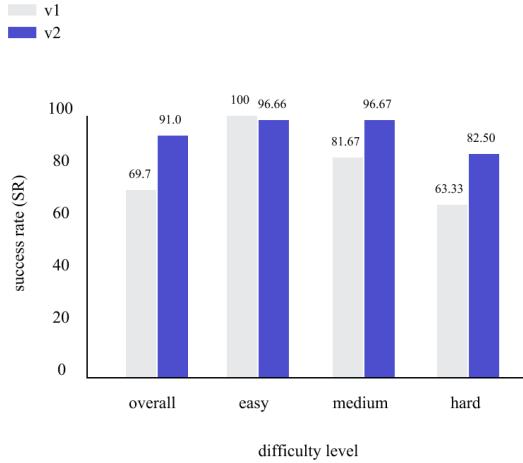


Fig. 2. Our approach (purple) compared to the best Success Rates from Table 4 in the original MedAgentBench paper (gray)

Task-level Highlights. Some tasks showed particularly notable gains. Task 6, which requires averaging vital signs, improved significantly due to the addition of a calculator tool that allowed the agent to compute aggregate values directly. Task 10, involving conditional ordering, benefited from refined conditional logic provided in the system prompt, enabling the agent to better interpret eligibility criteria based on patient age (see Appendix A.1).

While these results mark a promising step forward, clinical settings demand agents that are not just strong but highly reliable. In practice, agents must operate with greater than 95% accuracy to ensure patient safety and avoid issues such as inappropriate medication orders, unnecessary cancellations, or delayed interventions.

3.2. Effects of Memory

Introducing memory led to significant improvements in agent performance. Overall success rate increased from 91% to 98%, with the most notable gains observed on more complex tasks. For tasks requiring three or more tool calls (Tasks 5, 6, 9, and 10), success rate rose to **96.67%**, driven in large part by the agent achieving 100% accuracy on Tasks 9 and 10.

Inspecting Task 9 Performance. Task 9 requires the agent to order potassium replacement only if the patient’s potassium level is low. In earlier attempts, the agent incorrectly issued a replacement order regardless of the value. After being exposed to memory, it understood that it should only act if potassium levels are below a certain threshold. Agent’s internal reasoning for task 9:

Reflection on potassium search results:

- The most recent potassium value for patient S6352985 is 4.1 mmol/L, measured on 2023-11-13T04:55:00+00:00.
- This value is above the threshold of 3.5 mmol/L, so the potassium is not low.

Instruction validation:

- The instruction states to order potassium replacement and a follow-up lab only if the potassium is low (<3.5).
- Since the potassium is not low, no replacement or follow-up lab is needed.

Final action:

- Return the most recent potassium value (4.1) as requested.

Proceeding to finish with the value.

Inspecting Task 10 Performance. Task 10 asks the agent to check the latest HbA1C result and order a new test if the value is more than a year old. Previously, the agent failed to act when no HbA1C result was present. After incorporating memory that explicitly explained how missing data should be treated, the agent correctly placed a new lab order. Agent's internal reasoning for task 10:

Reflection on HbA1C search results:

- There are no HbA1C values for patient S2703270.

Instruction validation:

- The instruction says to order a new HbA1C test if the last result is more than 1 year old or if there are no lab results. \\

- Since there are no lab results, a new test should be ordered.

Final action:

- Return [-1] as requested.

Proceed to place an order for a new HbA1C lab test.

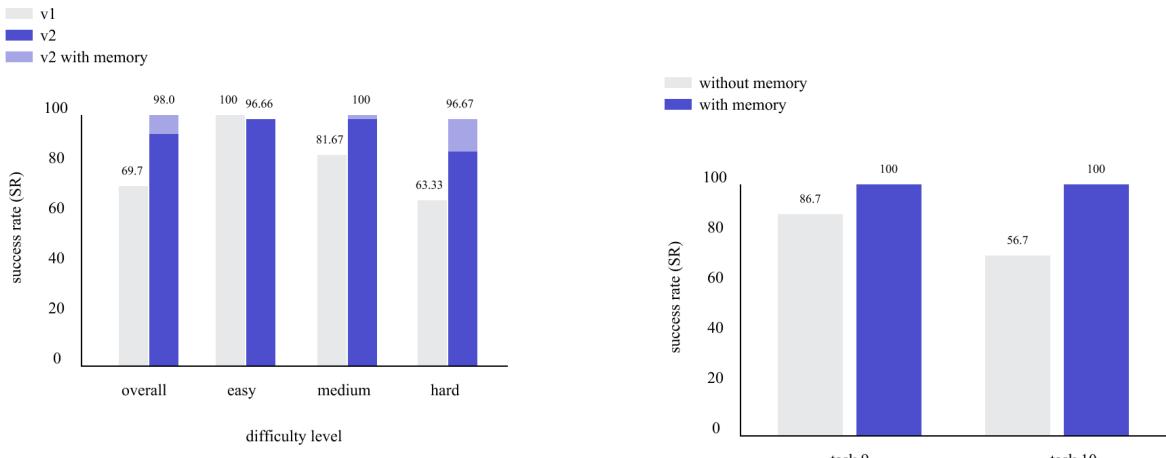


Fig. 3. Previous MedAgentBench implementation from the original paper (v1) compared to our approach before and after adding the memory component

Task-Specific Learning. When the agent receives explicit feedback through memory, it can successfully correct prior mistakes on the same task. By recalling the correct reasoning path, the agent re-applies the logic when encountering that task again. This shows that memory

Fig. 4. Tasks 9 and 10 success rate using our approach with memory

serves as an effective mechanism for instruction-specific refinement, improving reliability on known failure modes.

Learning the “Vibe” of Instructions. Perhaps most surprising is that memory not only corrected specific mistakes, but also led to broader improvements in instruction understanding. After receiving feedback on one task, the agent performed better on a different task that had no associated memory entry.

After running the memory train/test split (Sec. 2.3), the agent synthesized a memory for Task 10 treating “no existing HbA1C result” as “ ≥ 1 year old” and placing a new HbA1C order, and appended it to the system prompt. When we then reran Task 9 with this updated prompt, the agent no longer misinterpreted the conditional and stopped pairing a follow-up lab when potassium was normal. Although the new memory was specific to HbA1C, it improved the agent’s broader parsing of conditional clauses and implied “pairing” semantics, indicating that targeted memories can reshape how instructions are bracketed and interpreted across tasks, not just the task that triggered the memory.

This unexpected result suggests that memory helps the agent internalize patterns in how users phrase instructions, such as conditional logic, implied thresholds, or underspecified expectations. In effect, the agent begins to develop a kind of “vibe reasoning,” learning to interpret intent even when instructions aren’t fully spelled out. Rather than retrieving memories specific to the task’s instructions, we provide the agent with all memories so it learns new tasks over time. This capability is critical for real-world clinical use, where directions are often brief, implicit, or inconsistently structured.

We find that understanding the style of the user’s prompt itself is part of contextual understanding. Users differ not only in what they ask, but in how they ask. A short, imperative style often carries implicit assumptions (“check labs” may mean “check only recent labs” for one clinician but “order new labs if none exist” for another). For example, the seemingly simple instruction “go run some labs” may signal very different actions depending on the speaker: ordering a full panel, rerunning only abnormal values, or checking the chart before proceeding. These stylistic and pragmatic cues shape interpretation as much as the literal content, and memory enables the agent to gradually adapt to these individualized patterns of instruction over time.

3.3. *New Task Performance*

After evaluating our agent on the original MedAgentBench tasks and observing improvements with memory integration, we assessed its performance on the newly designed tasks to test how the agent generalizes to unseen instructions. On the new 300 task set, the agent achieved an overall success rate of **88.67%** (266/300).

Performance remained consistent with the original benchmark, indicating that the agent’s reasoning and decision-making capabilities generalized well to new clinical scenarios. Despite increased complexity and more decision steps in the new tasks, there was no significant drop in success rate, suggesting that the agent remains effective across different task formats and instruction wording.

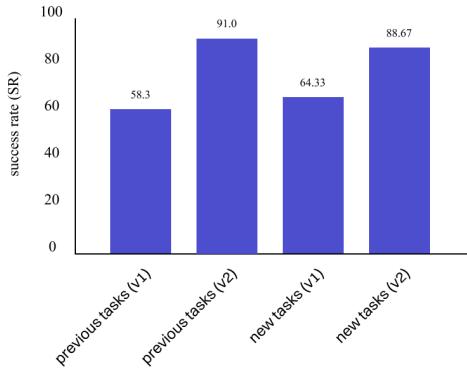


Fig. 5. Success rate (SR) of MedAgentBench’s agent (v1) compared to our approach (v2) on GPT-4.1 on previous tasks and new tasks

3.4. Common Error Analysis

We examined recurring failure modes on the new 300-task set to understand where the agent’s reasoning or EHR operational knowledge breaks down.

Understanding New Medical Terminology (Task 2). The task required identifying duplicate heparin orders and discontinuing the older one.

The agent did not recognize that discontinuing an order simply requires issuing a `MedicationRequest` resource with the same medication but with a status of `canceled`’ or `completed`. This limitation suggests that, while the agent is capable of following explicit instructions, it struggles with more implicit or abstract directives that require operational familiarity with EHR conventions. Providing additional clinical case examples, especially those involving indirect or context-dependent actions, may help the agent learn how to translate less explicit instructions into appropriate EHR interactions and improve operational familiarity with how actions are represented in FHIR.

Medication normalization and partial coding (Task 7). When asked to discontinue QT-prolonging medications, the agent canceled one ondansetron order but missed another whose EHR entry had no code and mentioned ondansetron only in free text. This shows the agent struggles when the same drug appears in different ways (coded vs. plain text) and needs to match names across codes and synonyms. In real use, the agent should present medication order creations and discontinuations as recommendations, with physicians making the final decision.

Format sensitivity in evaluation (Task 3). The task asked for the average heart rate over the past 6 and 12 hours. For patients with no qualifying data, the agent returned a single “-1,” while the evaluator expected a pair (-1, -1). The clinical logic was correct, but the response shape was misformatted, leading to an automatic “incorrect” label. This highlights that minor schema mismatches (types, cardinality) can obscure otherwise correct reasoning and should

be explicitly constrained in task specifications and outputs.

3.5. *Time-Cost Analysis*

In real-world clinical settings, agent design should weigh success rate (SR) together with time and cost, because these three determine whether large task queues can be handled within budget and service-level limits. Longer prompts, verbose plans, broad retrievals, and large memory blocks increase token usage and therefore unit cost, while slower responses extend latency; both effects compound at scale (see Appendix A.4). The core trade-off is straightforward, by example: a higher-SR agent that is slower and more expensive may raise the cost per successful task and delay throughput, whereas an agent that is 1% lower on SR but much cheaper and faster may clear more work for the same spend, at the expense of slightly more residual errors that require oversight. Healthcare institutions should analyze SR jointly with per-task cost and latency so that time and cost are evaluated alongside accuracy when determining real-world scalability.

4. Discussion

Our findings demonstrate that high-performing clinical agents are within reach. Our agent achieved strong performance not only on the original MedAgentBench tasks but also on a new set of clinically relevant tasks that were more complex and required broader reasoning across multiple EHR domains. This suggests that with careful design, LLM-based agents can begin to generalize to more clinical use cases.

However, better performance on existing benchmarks is not the end goal. Current benchmarks, while valuable, often lack the complexity, diversity, and clinical specificity needed to truly stress-test autonomous agents. To meaningfully advance the field, we need a new generation of benchmarks that are more diverse, contain more steps, use richer clinical terminology, and mirror the nuance of real patient care. To that end, we added 300 new tasks, expanding the benchmark with increased variety and difficulty. But this is only a starting point. If we want truly autonomous clinical agents, we must continue to push the boundaries of what we evaluate.

We also demonstrated early signs of how context engineering can improve agent performance and push towards deploying more reliable agents. The introduction of a memory mechanism that allows agents to adapt based on prior errors. By recording mistakes along with correct behavior and feeding it back into the agent's system prompt, we enabled the model to not only fix its past failures but also improve performance on other tasks. This capability, learning from feedback and applying it beyond a single instance, is a foundational requirement for agents that operate in dynamic, real-world clinical settings.

Beyond benchmark results, real-world integration raises operational and safety concerns. Implementing a memory-based approach requires physician oversight to verify that the agent is learning the correct behavior over time and not amplifying spurious heuristics; memories should be auditable, versioned, and attributable to specific corrections. Institutions will also need clear guardrails around scope of automation, human review of consequential actions (e.g., medication orders), sandboxed and logged EHR interactions, and monitoring for drift or

regressions. Finally, integration must align with institutional risk management and applicable regulatory review processes, including documentation of intended use, validation against representative clinical workflows, post-deployment monitoring, and incident reporting. These practical constraints shape not only whether an agent is accurate, but whether it is deployable in routine care.

4.1. *Limitations*

Task Design. When developing benchmarks to evaluate clinical agents, it is important to analyze how tasks are constructed and whether they realistically reflect the constraints of clinical settings. In the case of MedAgentBench, we observed that ambiguous or under-specified instructions frequently led to agent failures, not necessarily due to reasoning flaws, but due to unclear expectations. Furthermore, the task instructions used in both MedAgentBench and our study were often overly explicit, providing detailed descriptions and well-structured conditional logic.

This does not fully mirror real-world clinical practice, where physicians may issue terse, implicit instructions and expect the system to infer missing context. In practice, clinicians are unlikely to write out full logical conditions or explain task dependencies in natural language; they expect systems to 'just understand.' Developing agents that can reliably interpret such underspecified instructions remains an open challenge and will be critical for enabling real-world deployment. As such, future benchmarks should aim to capture this mismatch and evaluate agents in settings where instruction quality varies and contextual inference is necessary.

Evaluation Design. In some cases, the agent arrived at a correct response, but outputted it in an unexpected format. As a result, the evaluation function, designed for expected response formats, flagged these responses as incorrect. This issue of rigid format matching was also observed in the original MedAgentBench evaluation approach, where correct outputs were marked incorrect due to being incorrect in structure. This highlights the need for more flexible and semantically aware evaluation mechanisms when outputs vary in formatting or expression.

4.2. *Future Work*

Context Engineering. As LLM agents become more capable, the bottleneck is no longer just model performance; it's whether the model has been given the right context to succeed. Our study shows that when provided with well-structured prompts, curated tools, and clear instructions, agents can reliably perform complex clinical tasks. However, in real-world deployments, such ideal inputs are rare. Bridging this gap requires more than prompt engineering; it requires context engineering.

Going forward we see context engineering as the 'last mile' problem in clinical AI: not a model change, but a system design challenge. Done well, context engineering can turn LLMs into trustworthy clinical assistants that learn, adapt, and reason with precision.

Additionally, we would like to see benchmarks that qualitatively assess how an agent performs on tasks with multiple correct answers, such as "get me the patient history". A

possible first step for medical AI's deployment could be getting relevant context for a physician to make diagnoses and aid with clinical decision making.

Code

Code is publicly available at: <https://github.com/ericoericochen/medagentbenchv2>.

Appendix A.

Appendix A.1. *System Prompt*

System Prompt

You are an expert medical AI agent.

You will be given a clinical task to perform that involves interacting with a FHIR-compliant EHR system.

Everything you need to complete the task is in the EHR. Do not ask any clarifying questions to the user.

Take your time and think through every step. You MUST plan extensively before each function call, and reflect extensively on the outcomes of the previous function calls.

You have access to the following tools:

- `fhir_patient_search`: search and filter for patients using FHIR search params
- `calculator`: evaluate mathematical expressions in python
- `fhir_observation_search`: search for observations for a patient by code
- `fhir_vitals_create`: file vital signs for all flowsheets
- `fhir_vitals_search`: search for vital signs
- `fhir_procedure_search`: search for procedures
- `fhir_condition_search`: search for conditions
- `fhir_medication_request_create`: create a medication request
- `fhir_medication_request_search`: search for medication requests
- `fhir_service_request_create`: create a service request
- `finish`: respond with the final answer in the correct data type

ALWAYS use the `finish` tool to respond with your final answer. The output format will be stated in the instructions or context.

You should always respond with an answer. IT IS IMPORTANT THAT THE TYPE OF ANSWER IS CORRECT. If a value is a number, DO NOT respond with the string version of it. There should not be empty responses like `[]`. Below are good vs. bad examples.

GOOD Examples:

```
finish({ value: [-1] })
finish({ value: ["S6330912"] })
finish({ value: [10] })
finish({ value: [5.5, "2023-11-13T10:15:00+00:00"] })
```

BAD Examples:

```
finish({ value: [] })
finish({ value: ["-1"] })
finish({ value: ["10"] })
```

<guidelines>

- Write a detailed step-by-step plan on how you would execute the task. **MAKE SURE TO INTERPRET THE INSTRUCTIONS CORRECTLY.**
- Always paraphrase and validate the instruction at the beginning of your plan, including identifying any conditional logic.
- Carefully interpret conditional phrases. For example, if an instruction says "If X, then do Y, and also do Z", treat both Y and Z as conditional unless Z is clearly independent.
- Do not perform any action unless all of its stated preconditions are satisfied.
- Validate every instruction before execution. Avoid assumptions.
- Supply all necessary parameters to search calls; the more specific, the better.
- Always use the calculator tool for math operations (e.g., addition, subtraction, dose).
- In your final response, respond only with the specific value requested. No units.
- Format dates as ISO strings.

</guidelines>

<memory> </memory>

You must be especially cautious about performing actions only when their preconditions are satisfied. Misinterpreting conditional statements can lead to clinically inappropriate or unnecessary actions.

Appendix A.2. *Memory Prompt*

Memory Augmentation Instruction

Add memory to the current_prompt. Since the current agent doesn't handle this task correctly, write instructions for a correct approach to the agent's memory so that when it sees the task again, it gets it right. Think about the task description, the agent's previous response, and what the evaluation function tests to figure out why the agent got the wrong response.

Use 1--3 sentences to correct its MAIN mistake. Start your response with:

when asked "...", ...

Example Response: when asked "If low, then order replacement IV magnesium according to dosing instructions.", low indicates a value below 1.5 mg/dL.

```
<task_description> {task_description} </task_description>
<agent_response> {agent_response} </agent_response>
<eval_output> {eval_result} </eval_output>
<current_prompt> {system_prompt} </current_prompt>
```

Appendix A.3. New Tasks

Table A1. Broad categories for new tasks

Broad Category	Example Instruction	Example Context
Imaging surveillance (<i>Tasks 1 & 5</i>)	"Confirm patient S6315806 has a malignant neoplasm of left kidney diagnosis. If true and the last CT Abdomen/Pelvis is more than 3 months old, order a new CT and referral to Interventional Radiology."	"The ICD-10 code for malignant neoplasm of left kidney is C64.2. The CPT code for CT Abdomen/Pelvis with IV contrast is 74177 (for oncologic follow-up). The CPT code for Interventional Radiology referral is CON417 with the request: 'Evaluate renal mass for biopsy or ablation.'"
Medication order validation (<i>Tasks 2 & 8</i>)	"Verify that every active opioid analgesic order for patient S6474456 has a matching naloxone prescription. If an opioid order is active without naloxone, create a naloxone order."	"Opioid medications to search for include hydromorphone, oxycodone, fentanyl, hydrocodone, morphine. The naloxone order is naloxone nasal spray 4 mg."
Observation data analysis (Task 3)	"Calculate the average heart rate over the past 6 hours and the past 12 hours for patient S6315806."	"The code for heart rate is 'HEARTRATE'. It's 2023-11-07T22:47:00+00:00 now"
Device monitoring (<i>Task 4</i>)	"Check if patient S6315806 has a urinary catheter that has been in place for more than 48 hours without a documented removal order. If so, create a service request to remove the catheter."	"The procedure code for urinary catheter is 'NUR1373'. It's 2023-11-16T10:00:00+00:00 now."
Clinical observation monitoring (<i>Tasks 6 & 7</i>)	"Retrieve the most recent TSH and free T4 values for patient S6474456. Based on the results, take appropriate action according to the protocol."	"The code for TSH is 'TSH' and for free T4 is 'FT4'. Protocol: If TSH \geq 10 uIU/mL on two separate instances of lab values (separated by at least one month in date of lab collection), start levothyroxine 25 mcg PO daily and order repeat TSH/FT4 in 6 weeks. If TSH is 4.5–10 uIU/mL and free T4 is normal (0.8–1.8 ng/dL), order repeat TSH/FT4 in 3 months (no medication yet)."
Preventive immunization management (<i>Tasks 9 & 10</i>)	"Determine the date of the last influenza vaccine for patient S6474456. If it was administered more than 365 days ago, order a new influenza vaccine for today."	"It's 2024-01-09T00:00:00+00:00 now. The CPT code for influenza vaccine is 90686. The vaccine to order is influenza vaccine (quadrivalent, preservative-free, IM) for today."

Appendix A.4. Time Cost Analysis

Table A2. Comparative analysis of token consumption, tool usage, latency, and cost across representative tasks executed with the gpt-4.1 model. Total Token Usage denotes the average number of tokens consumed per task, capturing both input and output tokens. Tool Calls indicates the average number of tool invocations performed by the agent during a single run; this value varies depending on whether the agent's instructions require additional reasoning or data retrieval. Latency reports the average end-to-end response time in seconds. Total Cost quantifies the average expense per task, calculated using the standard OpenAI pricing model based on input and output token usage.

Task	Total Token Usage	Tool Calls	Latency (sec)	Total Cost (\$)
Task 1	6,291	2	7.805	0.0141
Task 2	10,068	3	11.560	0.0222
Task 3	10,791	3	18.254	0.0249
Task 4	14,766	2.2	20.632	0.0237
Task 5	10,937	2.4	20.524	0.0247
Task 6	10,074	2	16.892	0.0232
Task 7	32,415	3	14.470	0.0667
Task 8	6,596	2	12.874	0.0157
Task 9	25,102	2.2	20.324	0.0535
Task 10	8,697	2.4	18.118	0.0206

References

1. Y. Jiang, K. C. Black, G. Geng, D. Park, J. Zou, A. Y. Ng and J. H. Chen, Medagentbench: A realistic virtual ehr environment to benchmark medical llm agents (2025).
2. H. S. Saag, K. Shah, S. A. Jones, P. A. Testa and L. I. Horwitz, Pajama time: Working after work in the electronic health record, *Journal of General Internal Medicine* **34**, 1695 (2019).
3. A. Pal, L. K. Umapathi and M. Sankarasubbu, Medmcqa : A large-scale multi-subject multi-choice dataset for medical domain question answering (2022).
4. D. Jin, E. Pan, N. Oufattolle, W.-H. Weng, H. Fang and P. Szolovits, What disease does this patient have? a large-scale open domain question answering dataset from medical exams (2020).
5. Q. Jin, B. Dhingra, Z. Liu, W. W. Cohen and X. Lu, Pubmedqa: A dataset for biomedical research question answering (2019).
6. R. K. Arora, J. Wei, R. S. Hicks, P. Bowman, J. Quiñonero-Candela, F. Tsimpourlas, M. Sharman, M. Shah, A. Vallone, A. Beutel, J. Heidecke and K. Singhal, Healthbench: Evaluating large language models towards improved human health (2025).
7. T. Kojima, S. S. Gu, M. Reid, Y. Matsuo and Y. Iwasawa, Large language models are zero-shot reasoners (2023).
8. T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, Language models are few-shot learners (2020).
9. P. Chhikara, D. Khant, S. Aryan, T. Singh and D. Yadav, Mem0: Building production-ready ai agents with scalable long-term memory (2025).