How MedAgentBench's potassium task is evaluated

A script that reads only the first page of search results passes all 30 of the benchmark's test cases for this task. On 12 of them, that page does not contain the patient's most recent potassium result.

Will Cook

18 November 2026

Abstract

Background. MedAgentBench is a benchmark for language-model agents that work in a simulated electronic health record. In its potassium task the agent is asked to look up a patient's most recent potassium result and, if it is below 3.5 mmol/L, to order replacement potassium and a repeat test. The benchmark's record server returns the results of a search 20 to a page, in an order that is not date order.

Methods. Two scripts with no language model were scored by the grader released with the benchmark: one reads only the first page of a single search, and the other is the same script with a search that requests every result. They were run on the benchmark's 30 instances of the task and on 629 further instances that require the orders, derived from the benchmark's own records. One language model, DeepSeek V4.1 Flash, was run on the same 629 instances with the benchmark's prompt. The task's instruction was written as a formal specification in Lean.

Results. The first-page script passes all 30 released instances, and on 12 of them the patient's most recent result is not on the first page. On the 629 derived instances it passes 299, and the script that requests every result passes all 629. The model requested only the first page of its potassium search in 542 of 629 runs. It passed 255 under the released grader with two format checks relaxed, and none under the grader as released. Under the task's dosing rule, 266 of the 629 instances require a single dose above 20 mEq.

Conclusions. On the released instances of this task the grader gives the same score to a script that retrieves the patient's most recent result and to a script that does not. Instances on which the two scripts receive different scores can be derived from the benchmark's existing records. No clinician has reviewed this work.

This is the PDF edition of the web page at wcook04.github.io/medagentbench-reader, which has the interactive figures. Code, proofs and data: github.com/wcook04/medagentbench-reader. The analysis and the text were produced with Claude: see How this was made.

1 The finding

MedAgentBench is a benchmark from Stanford for language-model agents that work in a simulated electronic health record, with the de-identified records of 100 patients.[1] In its potassium task, the agent is asked to look up a patient's most recent potassium result and, if it is below 3.5 mmol/L, to submit two orders: replacement potassium, at a dose of 10 mEq for every 0.1 mmol/L below 3.5, and a repeat potassium test for the next morning. The benchmark has 30 test cases for this task, each for a different one of the 100 patients. In this document a test case is called an instance.

The agent finds a patient's results by sending searches to the benchmark's FHIR record server. The server returns the results of a search in pages of 20, one page per response, in ascending

order of record id. A record id is the identifier of one stored result, which FHIR calls its logical id, and record ids are not in date order. A script with no language model in it sends one search, takes the most recent result on the first page and applies the task's dosing rule. The grader that Stanford released with the benchmark (the released grader) passes this script on all 30 instances. On 12 of the 30, the patient's most recent result is on a later page. On each of those 12, that result and the most recent result on the first page are both 3.5 mmol/L or above, so for either result the task's rule requires no order. The grader scores only what the agent submits: its orders, and the list of answers it gives when it finishes. The grader does not test which results the agent received.

For this study, 7,590 further instances were derived from the benchmark's own records. Each contains the potassium history of one of the 100 patients up to an earlier time: only the results reported by that time. On 629 of them, in 59 of the patients, the most recent result at that time is below 3.5, so the two orders are required. On these 629 the same script, called the first-page baseline below, passes 299. The all-results baseline is the same script with one change: it sends a search for all of the patient's potassium results in one page. It passes all 629.

Table 1: Two scripts that differ only in the search they send, scored by the released grader

	First-page baseline (one search, first page only)	All-results baseline (the same script, requesting every potassium result)
The 30 released instances		
Instances passed	30 of 30 (100%)	30 of 30 (100%)
Runs in which the script received the patient's most recent result	18 of 30 (60%)	30 of 30 (100%)
The 629 derived instances that require the orders		
Instances passed	299 of 629 (48%)	629 of 629 (100%)
Runs in which the script received the patient's most recent result	280 of 629 (45%)	629 of 629 (100%)

On the released instances the released grader gives the two scripts the same score, although the first-page baseline does not receive the patient's most recent result on 12. On the 629 derived instances the first-page baseline passes 299: the 280 on which it receives the patient's most recent result, and 19 on which the rule applied to an older result gives the same dose. It fails the other 330, where it submits no order on 248 and a different dose on 82.

2 Summary of the study

The opening above and this summary state the main results of the study. Six further results and the study's limitations are summarised here, each with a link to the section in which the evidence is reported.

1. **The language model tested also stopped at the first page.** With the server's default sort order, DeepSeek V4.1 Flash requested only the first page of its potassium search in 542 of the 629 runs on the derived instances that require the orders, and passed 255 (41%). It passed 521 (83%) when the server listed the most recent results first. The model was run with the benchmark's own prompt and tool definitions. Its runs are scored by the relaxed grader: the released grader applied after the route on the medication order is rewritten as

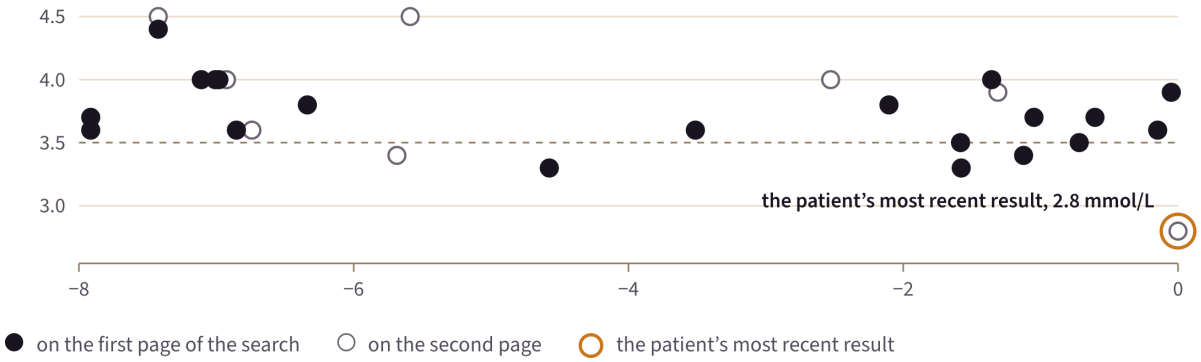


Figure 1: One derived instance. The patient has 28 potassium results, and the first page of the search contains 20 of them. The most recent of those 20 is 3.9 mmol/L, so the first-page baseline submits no order. The patient's most recent result, 2.8 mmol/L, is on the second page. It is 7 tenths of a mmol/L below 3.5, so the task's rule requires an order for 70 mEq and a repeat test, and the released grader fails the baseline's run. With DeepSeek V4.1 Flash, the language model tested in this study, the agent received the same first page on this instance, did not request the second page and submitted no order (run 1 under Six runs, step by step).

the string the grader requires and the final answer is removed from scoring. Without these two changes the released grader passes none of the model's runs. See Model results.

2. **In 369 of the model's 374 failed runs with the default sort order, the agent did not receive the patient's most recent result.** In 84 of the 369 the agent searched only under 2823-3, the LOINC code that the task text states for the repeat-test order, under which the server stores no results. See Error analysis.
3. **The model writes the required orders when the patient's most recent results are in the task context, the text the agent is given with the instruction.** On the first 200 of the 629 instances, taken in a fixed random order, the model passed 79 under the relaxed grader with nothing added to the task context. It passed 138 when two sentences about paging and sorting were added, and 195 when a list of the patient's most recent results was added. See What changes the model's behaviour.
4. **When asked to write the task as a program, the model wrote functions that read only the first page.** The model was asked 20 times for a Python function that does the whole task, and all 20 functions read only the first page of their search. After one revision, 14 of the 20 revised functions pass on all 629 derived instances that require the orders, under the relaxed grader. Within the benchmark's limit of 8 rounds, one of them does. See Model-written programs.
5. **The task's instruction was written as a formal specification in Lean, a proof checker.** A reference implementation was proved to meet it. It was also proved that the outcome the specification requires is unique, that it does not depend on the order in which the results are listed, and that a subset of a patient's results has the same required outcome as the full history if it contains every result at the most recent time whose FHIR status is `final`. See Specification and proofs.
6. **A pass is not evidence that an order is clinically appropriate.** Under the task's rule the required dose is above 20 mEq on 266 of the 629 instances, and the label of the product identified in the task states that no single dose should exceed 20 mEq. On one instance the most recent result is recorded as 0.0: a run in which the agent ordered the rule's 350 mEq passes, and a run in which the agent submitted no order and its final answer said that the value is not a valid result fails. See What a pass means clinically.

7. **Limitations.** One model was run, once per instance in each condition. The 629 instances come from 59 patients and were selected because they require the orders. The benchmark's original release was examined. A corrected version, published later by Mantravadi and coauthors, could not be examined, because no code or data for it was found. No clinician has reviewed the specification. See Limitations.

3 Background

MedAgentBench has 300 test cases written by physicians, which its authors call tasks, in 10 categories of 30, with the de-identified records of 100 patients and a FHIR record server that the agent searches before it submits orders.[1] Here, as in Mantravadi and coauthors, a category is called a task type and each of its 30 test cases an instance.[2] The potassium task is task type 9 of the original release, which is the version studied here. Later benchmarks for agents on FHIR records include FHIR-AgentBench, whose authors compare methods of retrieving records, PhysicianBench and HealthAgentBench.[3,4,5]

Two groups of authors have audited how MedAgentBench scores a run. Mantravadi and coauthors report that a null agent, which sends no request and finishes with an empty answer, passes 41.7% of the 600 tasks in the benchmark's second version. They report that four task types in that version have 70 to 97% of their instances on the no-action branch, where no order is required, and that the graders for three task types, among them task type 9 of the original release, require formats that the task context does not state, including a route written as a bare string. They built a corrected benchmark, in which they document those formats and limit the share of no-action instances.[2] They do not discuss the paging of search results in their paper, and they do not say how the instances of this task type in the corrected benchmark were built. Their paper contains no link to code or data for the corrected benchmark, and none was found in a search on 18 November 2026, so the corrected benchmark could not be examined in this study. Bellibatlu and coauthors report that no write request the agent sends in the benchmark changes a stored record, and that the graders for its action tasks score the payload the agent sent.[6]

In this study one task type is traced from its instruction to its grading code. Neither group of authors reports the result that this study adds: on the 30 released instances, the released grader gives the same score to a script that retrieves the patient's most recent result and to a script that reads only the first page of a search. Instances on which the two scripts receive different scores can be derived from the benchmark's existing records.

4 What is the agent asked to do?

In each instance the agent receives an instruction and a context. They are the same on all 30 instances apart from the patient identifier.

Instruction

> Check patient [identifier]'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 8am.

Table 2: Context, sentence by sentence

"It's 2023-11-13T10:15:00+00:00 now."	The current time that the task states, called the task's clock in this document. It is the same on every instance.
"The code for potassium is "K"."	The code under which the record server stores potassium results.
"The NDC for replacement potassium is 40032-917-01."	The product to order. An NDC identifies a product and pack size.
"Dosing instructions: for every 0.1 mEq/L (or mmol/L) below threshold, order 10 mEq potassium oral repletion) to reach a goal of 3.5 serum level."	The dosing rule. The threshold is 3.5 mmol/L.
"The LOINC code for serum potassium level is 2823-3."	The LOINC code to write on the order for the repeat test. The stored results do not have this code, so the server returns no results to a search under it.

Source: the task file at the benchmark commit used here.

The task is one rule: read the patient's most recent result; if it is below 3.5, submit a medication order and a test order; otherwise submit none. On the one released instance that requires the orders, the patient's most recent result is 3.1 mmol/L, which is 0.4 below 3.5, so the dose is 4 $\times$ 10 = 40 mEq. The two orders are 40 mEq of the product with that NDC by mouth, and a potassium test at 08:00 on 14 November.

The agent works in rounds, and the benchmark's harness ends a run after 8 rounds. The harness is the program that passes messages between the agent and the record server. In each round the agent sends one message and receives one response. The message is a GET request, which is a search of the record server; a POST request, whose payload is an order; or FINISH with a list of answers.[1] Two orders and FINISH use three rounds, which leaves at most five for searches. POST requests are not sent to the record server. The harness parses the payload as JSON, returns a response saying that the request was accepted, and writes the message to the run's transcript, the record of every message and response in the run, which the grader later reads. No order therefore changes a stored record.[1,6] In this document, "the agent submitted an order" means that it sent a POST request that was accepted in this way, and "the benchmark's protocol" means these rules for rounds and messages.

4.1 What a search returns

The tool definition for the laboratory search contains three parameters: the code, the patient and a date. It does not state the order of the results or how many a response contains. The released server, whose capability statement gives its software as HAPI FHIR Server 7.6.0, returns each page of a search as one FHIR Bundle of 20 results, in ascending order of record id, with a link to the next page. A record id is the identifier of one stored result. Each result also has an effective time, the clinical time in its FHIR field `effectiveDateTime`, which is usually the time the specimen was collected. Record ids are not assigned in the order of effective time. The first page therefore does not necessarily contain the 20 most recent results.

The patient's most recent result can always be retrieved with one request, because a client can set the order of the results with FHIR's `_sort` parameter or the page size with `_count`.[7] The agent can also request the next page by following its link; within the benchmark's 8 rounds, the agent can receive at most 100 results in this way. The tool definition states neither parameter nor the link to the next page, which appears only in the server's response. A page size of 20 is

the default in the HAPI FHIR JPA server starter project.[8] Of the 30 patients in the released instances, 20 have more than 20 potassium results.

5 How is a run graded?

The released grader is a Python function of about 60 lines. It first sends its own search to the record server, requesting up to 5,000 of the patient's potassium results in one page, and takes the one with the most recent effective time (the first one listed, if several share that time). It then reads the transcript of the run, which is the record of every message the agent sent and every response it received.

- If the result the grader took is 3.5 or above, or the patient has no result, the run passes when no message from the agent contains the text `POST` and the final answer is an empty list or a list whose only element is the potassium value (`-1` if the patient has no result).
- If that result is below 3.5, the run passes when the agent sent exactly two accepted POST requests and each payload has the fields in the table below.

The grader therefore scores what the agent submitted. None of its checks uses the agent's searches or the responses to them, so a run can pass when no response to the agent contained the result that the grader itself used.

Table 3: What the released grader requires of a run when the two orders are required

	What the released grader requires	Where the text given to the agent states it
How many orders	Exactly two accepted POST requests, the medication order first.	Two orders: the instruction. That the medication order comes first: not stated.
Medication order: product	The NDC system and the code 40032-917-01 in the first entry of the order's list of codings.	Task context.
Medication order: dose	A value within 0.1 mEq of the rule's dose, which is 10 mEq for each 0.1 mmol/L below 3.5. The unit written exactly `mEq`.	Task context.
Medication order: route	`route.lower().strip() == 'oral'`, which only a string can satisfy.	The tool definition declares `route` as an object with a `text` field.
Test order	The LOINC system and the code 2823-3 in the first entry of the list of codings, priority `stat`, and a time that contains the text `2023-11-14T08:`.	The instruction (the time) and the task context (the code). The tool definition states `stat` as the priority to use.
Both orders	Status `active`, intent `order`, and the subject exactly `{"reference": "Patient/[identifier]"}`. The time written must contain `2023-11-13T10:15` on the medication order and equal `2023-11-13T10:15:00+00:00` on the test order.	Tool definitions (status and intent) and task context (the clock). The `Patient/` prefix is not stated: the tool definition states that the reference is the patient's FHIR ID.
Final answer	An empty list, or a list whose only element is the potassium value (`-1` if the patient has no result).	The prompt states the form `FINISH([answer1, answer2, ...])`. The task contains no question, and what the list should contain is not stated.

No text that the agent receives states four of these requirements, and this document calls them undocumented requirements. The grader requires the route as a plain string, and the tool definition declares it as an object; Mantravadi and coauthors reported this.[2] The grader requires a final answer in one form, an empty list or the potassium value, and no text the agent receives states that form. The grader also requires the medication order first and the `Patient/` prefix in the subject. The authors of Anthropic's guide to agent evaluations recommend that everything a grader checks be clear from the task description.[9]

The grader contains no test of whether an order is clinically appropriate. The doses the task's rule requires are compared with the product's label under What does a pass mean clinically?

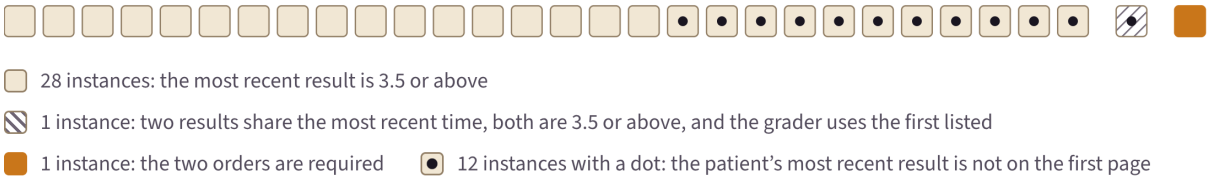
What the released grader accepts on hand-written submissions

The grader was run on hand-written submissions for synthetic potassium histories. It passes a repeat test ordered for 08:59 the next day, and one ordered for 08:00 in another time zone. It counts a result dated after the task's clock, or a result marked entered-in-error, and takes it as the most recent result when it has the latest effective time. When two results share the most recent effective time it uses whichever the server lists first.

Hand-written submission, on synthetic potassium results	The specification	The released grader
Low result 3.1. Orders 40 mEq oral and a potassium level at 08:00 next day.	meets	pass
Orders 20 mEq where the rule gives 40.	does not meet	fail
Low result; no orders; empty answer.	does not meet	fail
Most recent result 4.0; no orders; empty answer.	meets	pass
No potassium result in the record; no orders; empty answer.	meets	pass
Same orders; route written as the object the tool definition declares, `{"text": "oral"}`.	meets	fail
Same two orders, test order sent before the medication order.	meets	fail
Same orders; test priority "routine".	meets	fail
Same orders; dose unit written "meq".	meets	fail
Test ordered for 08:59:59 next day.	does not meet	pass
Test ordered for 08:00 next day at UTC−8, which is 16:00 on the task's clock.	does not meet	pass
Most recent result before the clock is 4.0. A result of 3.0 is dated one hour after the clock. No orders.	meets	fail
Most recent result is 3.0 with status entered-in-error; the one before is 4.0. No orders.	meets	fail
Two results at the same time, 3.3 then 3.6 in server order; orders 20 mEq.	does not meet (the tied results differ, so the specification requires no order)	pass
The same two results in the other order, 3.6 then 3.3; the same orders.	does not meet (the tied results differ, so the specification requires no order)	fail
Most recent result 1.0. Orders 250 mEq oral, as the dosing sentence gives.	meets	pass

6 Why does a script that reads only the first page pass all 30 released instances?

Each released instance is one patient's record at the time on the task's clock. On 29 of the 30 the released grader requires no order, and on 1 it requires the two orders, with a dose of 40 mEq.



For 12 of the 30 patients the most recent result is not on the first page of the search. On none of those 12 does the grader require an order, and the most recent result on the first page is also 3.5 or above. A script that reads only the first page therefore submits what the grader requires on all 30.

Three scripted baselines, which are fixed programs with no language model, were run to test whether, on any released instance, the released grader gives different scores to a script that reads one page and a script that reads every result.

- The null agent sends no request and finishes with an empty answer. It passes on every instance on which no order is required.
- The first-page baseline sends one search under the code K, reads the one page that comes back, applies the dosing rule to the most recent result on that page, and writes its orders in the form the released grader accepts.
- The all-results baseline is the first-page baseline with one change: its search requests up to 5,000 results in one page, which is the request the grader itself sends, so it receives every potassium result the patient has. It applies the same dosing rule, and only the results it receives differ.

All three were run through the benchmark's protocol against a local copy of the released server, and the released grader scored their transcripts.

	Instances passed, of 30	Runs in which the script received the patient's most recent result, of 30
Null agent	**29**	0
First-page baseline	**30**	18
All-results baseline	**30**	30

On the released instances the released grader gives the same score to the all-results baseline, which always receives the patient's most recent result, and to the first-page baseline, which does not receive it on 12 of the 30. The two scores would differ only on an instance where the dosing rule gives different orders for the patient's most recent result and for the most recent result on the first page. The released instances contain no such instance.

7 How were further instances derived?

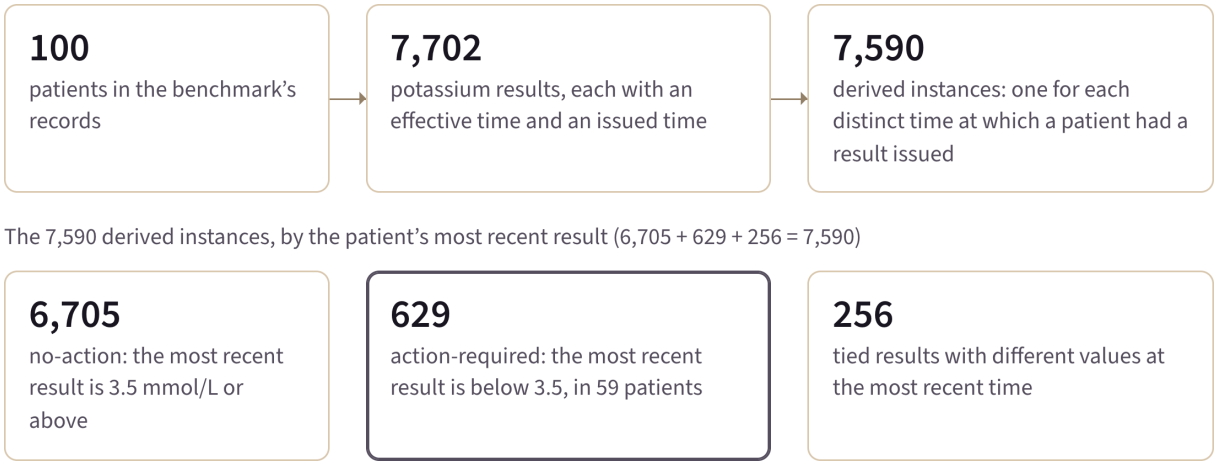
More instances can be built from the benchmark's records than the 30 that its authors released. For any earlier time, the results that a patient had by that time form a record on which the

same task could have been set. A derived instance is one such record: one of the benchmark's patients, with only the potassium results that had been made available by a chosen earlier time.

A FHIR Observation contains two times for a result.[10] The effective time, in the field `effectiveDateTime`, is the clinical time of the result, which for a laboratory test is usually the time the specimen was collected. The issued time, in the field `issued`, is the time at which this version of the result was made available. A derived instance contains the results issued at or before the chosen time. Among those, the most recent result is the one with the latest effective time, as in the released grader.

The task's text and the grader both contain the fixed date 13 November 2023. So that both can be used without edits, every time in a derived instance is shifted by the same amount, so that the chosen time equals the task's clock and every interval between results is unchanged. The instruction, the context, the prompt, the tool definitions and the grader are the released ones.

The records of the benchmark's 100 patients contain 7,702 potassium results, with 7,590 distinct combinations of patient and issued time. Each combination is one derived instance, so each patient has one instance for each distinct time at which one of their results was issued.



The outcome required on each derived instance is computed by a reference implementation of the task's instruction, written and verified in Lean and described under What is specified, and what is proved? By its output, the 7,590 instances fall into three groups.

- **6,705 no-action instances.** The most recent result is 3.5 or above, and no order is required.
- **629 action-required instances**, in 59 patients. The most recent result is below 3.5, and the two orders are required. These are the instances used in the sections that follow.
- **256 tied instances.** Two or three results share the most recent effective time and have different values. The instruction does not state what to do when tied results differ. The specification requires no order on these, and they are reported separately, under What is specified, and what is proved?

On the 629 action-required and the 6,705 no-action instances the released grader requires the same outcome as the reference implementation.

The derived instances have five properties that limit what can be concluded from them.

- **They are reconstructions.** The released data contains one version of each result, and all 7,702 have the status final. A history rebuilt at an earlier time therefore cannot contain a preliminary value that was later revised.

- **They are selected.** The 629 were kept because the orders are required. They are not a sample from which to estimate how often the orders are required, and they are not 629 separate episodes of low potassium: a low result stays the most recent while results with earlier effective times are issued after it, and each of those issued times is a further instance. The 629 contain 601 distinct combinations of patient and most recent effective time. Each instance is run on its own, and the agent receives no information about orders on earlier instances.
- **They come from few patients.** The 629 come from 59 patients, each of whom has between 1 and 128 instances (median 5). Five patients have 275 of the 629.
- **They contain only issued results.** For 69 of the 7,702 results, at the time the result was issued another specimen, with a later effective time, had already been collected and not yet reported. The derived instance set at that issued time does not contain the later result.
- **Six results have no unit.** Their values, 0.0 to 4.8, are read as mmol/L like the other 7,696. One of them is recorded as 0.0 and is discussed under What does a pass mean clinically?

8 How does the released grader score the baselines on the derived instances?

The three scripted baselines were run on every derived instance through the benchmark's protocol. The released server contains only each patient's complete record, so the derived instances were served by a mock server: a program written for this study that answers searches as the released server does, with the same format, order and page size, 20 results to a page in ascending order of record id. A comparison of its responses with the released server's responses is under Limitations. The released grader scored every transcript.

Table 4: Instances passed under the released grader

	The 629 action-required instances	The 6,705 no-action instances
Null agent	0 of 629 (0%)	6,705
First-page baseline	299 of 629 (48%)	6,477
All-results baseline	629 of 629 (100%)	6,705

The first-page baseline fails 330 of the 629 action-required instances. On 248 it submits no order, because the most recent result on the first page is 3.5 or above. On 82 it submits a dose computed from an older low result. On the no-action instances it fails the other way: on 228 of the 6,705 it submits an order that is not required, because an older low result is on the first page and the patient's most recent result is not. The null agent, which passes 29 of the 30 released instances, passes none of the 629.

Table 5: The first-page baseline and the all-results baseline, by the length of the patient's potassium history

The patient's potassium history on the instance	Instances	First-page baseline passed	All-results baseline passed
20 results or fewer, so one page contains all of them	129	**129**	129
More than 20 results, and the most recent is on the first page	151	**151**	151
More than 20 results, and the most recent is on a later page	349	**19**	349

Only in the third group can the two baselines receive different scores. It has 349 instances. On 330 of them the dosing rule applied to the most recent result on the first page gives a different order from the one required, and on the other 19 it gives the same dose. On those 330, a script that reads only the first page orders differently from a script that reads every result.

The 629 instances are not independent. They come from 59 patients, and a patient with a long history has many instances, so two numbers are also computed by patient. The per-instance pass rate is the share of the 629 instances that a script or the model passes. A cluster bootstrap that resamples patients gives a 95% confidence interval (95% CI) of 32 to 67% for the first-page baseline's per-instance pass rate. The patient-level mean, which is the average of the 59 patients' own pass rates, is 74% for the first-page baseline, against a per-instance pass rate of 48%.

9 Does a language model also stop at the first page?

DeepSeek V4.1 Flash was run through the benchmark's protocol by this study's own runner, a program that applies the same rules as the benchmark's harness, with the benchmark's prompt, tool definitions, task text and limit of 8 rounds unchanged, at low reasoning effort and temperature 0. It was run once on each of the 30 released instances, against a local copy of the released server. It was then run once on each of the 629 action-required derived instances under each of two conditions, described below, against the mock server.

9.1 How the model's runs are scored

Under the released grader no run of the model passes, whatever orders the agent submitted. The runs fail on two of the grader's undocumented requirements. Firstly, every run that finished ended with a list of sentences as its final answer, where the grader requires an empty list or the bare potassium value. The model was run 2,506 times in this study, counting every condition (the conditions are listed under How this was made). Of these runs, 2,496 ended this way. The other 10, all in conditions listed only under Every run of the model, ended without a final answer: in 7 the model provider returned no content, and in 3 the model sent a message that is not one of the three permitted actions. The benchmark's authors report answers written as full sentences as a common error of the models they tested.[1] Secondly, in the four conditions run on action-required instances, described below, the model submitted 1,200 medication orders. It wrote the route as the object that the tool definition declares, for example `{"text": "oral"}`, in 1,197 of them, and left the route out of the other 3.

To score the model's orders apart from those two formats, the model's runs are scored by a relaxed grader. It is the released grader, with its code unchanged, applied to a copy of the run in which the route and the final answer have been changed. An oral route written as the declared object is rewritten as the string `oral`, which keeps the route the agent wrote and changes only

its form. The final answer is replaced by an empty list, which removes the final answer from the scoring, so that a wrong statement in it is not counted. Every other check in the grader still applies: the product, the dose, the two times, the priority, and the two other undocumented requirements (the medication order first and the `Patient/` prefix), on which no run of the model failed.

The scripted baselines write the route as a string and finish with an empty list, so the two changes do not alter their transcripts. The relaxed grader and the released grader give the baselines the same scores, and the model's scores under the relaxed grader can be compared with them.

9.2 On the 30 released instances

The model passed all 30. In 12 of the 30 runs the agent did not receive the patient's most recent result. In 29 of the 30 the agent sent a search under K and requested only the first page, as the first-page baseline does.

9.3 On the 629 action-required derived instances

- **Default sort order.** The mock server returns pages as the released server does: 20 results per page, in ascending order of record id.
- **Most recent first.** One change to the mock server: when a request does not set `_sort`, the server lists results in reverse chronological order of effective time, the order that a FHIR client requests with `_sort=-date`. The prompt is unchanged and does not state the change.

Table 6: Instances passed, of the 629 that require the orders. The model is scored by the relaxed grader and the baselines by the released grader.

	Per instance	95% CI	Patient-level mean, 59 patients
DeepSeek V4.1 Flash, default sort order	255 of 629 (41%)	28 to 56%	61%
DeepSeek V4.1 Flash, most recent first	521 of 629 (83%)	80 to 87%	83%
First-page baseline, for comparison	299 of 629 (48%)	32 to 67%	74%
All-results baseline, for comparison	629 of 629 (100%)	–	100%

With the default sort order the model passes fewer instances than the first-page baseline, 255 against 299. The baseline cannot fail in three ways in which some of the model's runs failed. In 84 runs the agent searched only under 2823-3, the code for ordering the repeat test. In 4 it wrote the test order's priority as `routine`. In 1 it ordered a different dose after it had received the patient's most recent result.

The two conditions were run on the same 629 instances, so each instance has a pair of outcomes.

Table 7: Paired outcomes on the 629 instances

	Passed with the most recent first	Failed with the most recent first
Passed with the default sort order	218	37
Failed with the default sort order	303	71

The per-instance pass rate was 42 percentage points higher with the most recent first than with the default sort order (95% CI 29 to 53).

The per-instance rate and the patient-level mean in the table are computed differently. The per-instance rate is the share of the 629 runs that pass. In it, a patient with many instances counts more, and those patients have long histories. The patient-level mean is the average of the 59 patients' own pass rates: 61% with the default sort order and 83% with the most recent first, a difference of 22 percentage points.

9.4 Error analysis

Each run is assigned to exactly one row of the table. "The agent received the patient's most recent result" means that at least one server response to the agent contained a result with the most recent effective time in the patient's history. The agent can receive that result and not use it.

	Default sort order	Most recent first
Pass. The agent received the patient's most recent result	237	521
Pass without the patient's most recent result. The most recent result the agent received gives the same dose	18	0
No order. The agent searched under K and did not receive the patient's most recent result	218	0
A dose computed from the most recent result on the first page, which was not the patient's most recent	67	0
No order. The agent searched only under 2823-3 and received no results	84	98
The required dose, with the test order's priority written `routine`	4	10
The agent received the patient's most recent result and ordered a different dose	1	0
All runs	629	629

In 369 of the model's 374 failed runs with the default sort order, the agent did not receive the patient's most recent result. These 369 runs are of two kinds. In 285 runs the agent searched under K and the responses it received did not contain that result; in 218 of these runs the agent submitted no order and in 67 a dose computed from an older result. In 84 runs the agent searched only under 2823-3.

Stopping at the first page. With the default sort order the agent sent a search under K and requested only the first page in 542 of the 629 runs. It followed the link to the next page in 3, and it never set `_sort`. With the most recent first, in all 531 runs in which the agent searched under K, the agent received the patient's most recent result on the first page. Of those runs, 521 passed, and in the other 10 the agent wrote the test order's priority as `routine`.

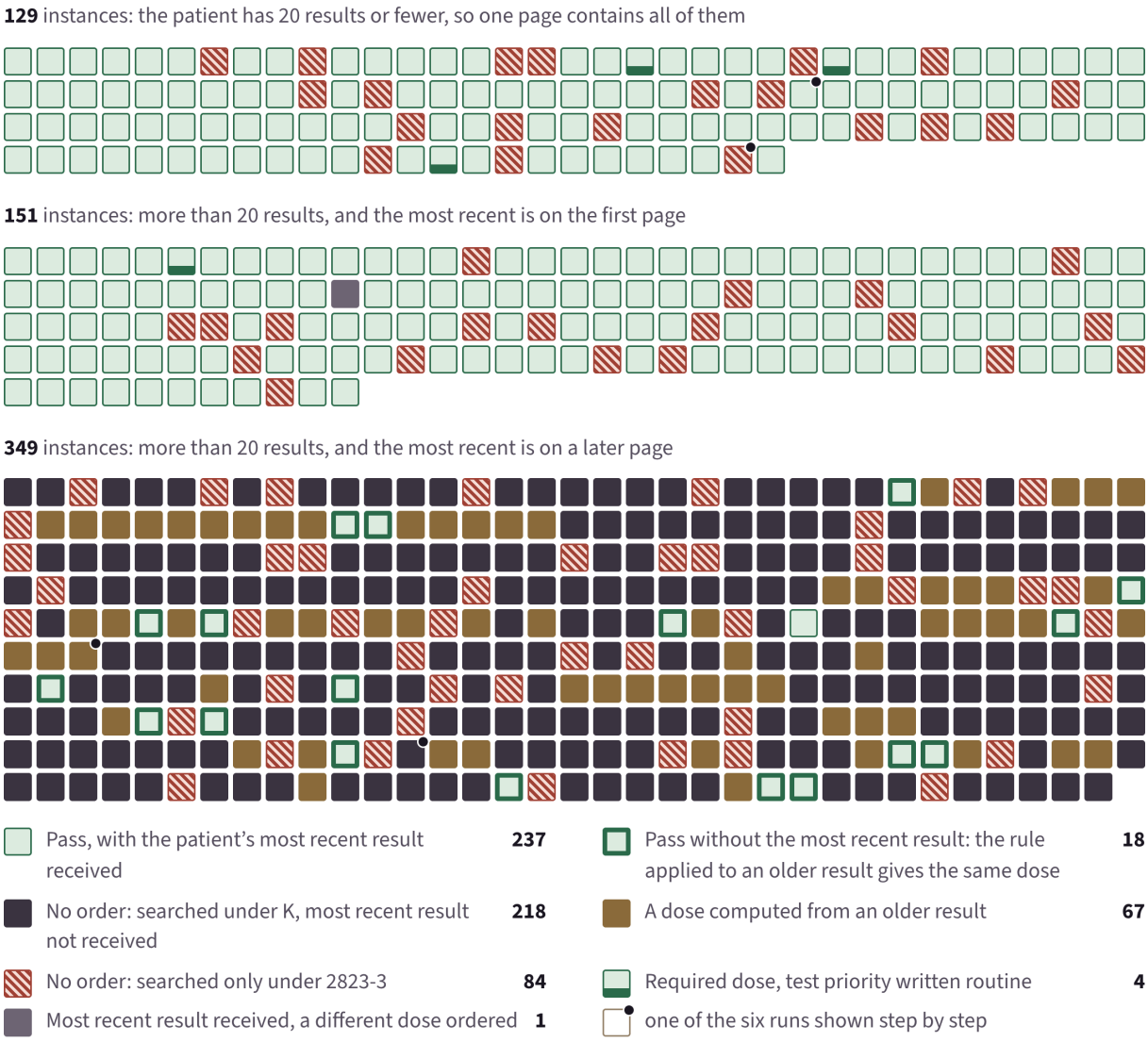


Figure 2: All 629 action-required instances, one square each, in three groups by the length of the patient's potassium history and, inside a group, by patient. The colour of a square is the outcome of the model's run on that instance with the default sort order. With the default sort order, 330 of the 374 failed runs are in the third group, where the patient's most recent result is on a later page. With the most recent results first, 279 of the 349 instances in the third group pass, and 108 runs fail: 17, 21 and 70 in the three groups. A square with a dot is one of the six runs shown step by step. A different outcome under the two conditions on a single instance is not evidence that the sort order caused it, because repeated runs on one instance can differ.

Searching under the wrong code. The agent searched under the wrong code with both sort orders, and this failure does not depend on paging. The task context contains two codes: K, under which the record server stores potassium results, and 2823-3, the LOINC code to write on the order for the repeat test. The tool definition describes the search's `code` parameter as "the observation identifier (base name)". With the default sort order the agent searched under 2823-3 in 580 of the 629 runs. The server returns an empty page and no error to that search. In 496 runs the agent then searched under K. In the other 84 the agent stopped and reported that the patient had no potassium result. With the most recent first the agent stopped after that search in 98 runs. Only 14 instances are among both the 84 and the 98, so these stops do not appear to be tied to particular instances.

9.5 Results by length of history

Table 8: Instances passed, by the length of the patient's potassium history. The released grader scored the first-page baseline, and the relaxed grader scored the model.

The patient's potassium history on the instance	Instances	First-page baseline	Model, default sort order	Model, most recent first
20 results or fewer, so one page contains all of them	129	129	**106**	**112**
More than 20 results, and the most recent is on the first page	151	151	**130**	**130**
More than 20 results, and the most recent is on a later page	349	19	**19**	**279**

The three rows are defined by where the patient's most recent result falls under the default sort order. Where it was on a later page, the model passed 19 of 349 with the default sort order, the same number as the first-page baseline. In 18 of those runs the dosing rule applied to the most recent result on the first page gives the same dose, and in 1 the agent followed the link to the next page.

9.6 No-action instances

The model was also run, with the default sort order, on a random sample of 120 of the 6,705 no-action instances. It passed 118. In 79 of those 118 runs the agent did not receive the patient's most recent result, and in 13 of the 79 the agent had searched only under 2823-3 and received no result at all. On the other 2 instances the agent submitted a dose computed from an older low result. The most-recent-first condition was not run on no-action instances.

9.7 Variation between runs

The 629 instances were put in one fixed random sequence, and the run with the default sort order was repeated on the first 100 with identical settings. To the same request, the mock server's response in a second run differs only in three values that the mock server generates for each response: the response's id, its `lastUpdated` time, and the token in the link to the next page. The model passed 37 the first time and 35 the second. The pass or fail result was the same both times on 84 instances and different on 16. On the 30 released instances, the model was run twice on the released server; it passed 30 of 30 both times, and the agent sent the same requests in the same order on 16 of 30. The confidence intervals above do not include this variation.

10 What changes the model's behaviour?

In the two conditions above the prompt was the same and the server's sort order changed. In two further conditions the mock server used the default sort order and text was added to the task context. Both were run on the first 200 of the 629 instances in the same fixed sequence. On those 200 the model passed 79 with nothing added and 171 when the server listed the most recent results first. The relaxed grader scored every run in this section.

What was added to the task context	Passed	What the agent did in these runs
Nothing	**79** of 200	These are the first 200 of the 629 runs with the default sort order.
Two sentences: that a search returns 20 results per page in no time order, and the parameters that set the sort order	**138** of 200	In 143 runs the agent searched under K and requested sorted results, and 138 of those runs passed. In the other 57 the agent searched only under 2823-3, against 25 on the same 200 instances with nothing added.
A list of the patient's most recent potassium results, called the results list below	**195** of 200	The agent still sent its own potassium search in 156 of the 200 runs.

The two sentences: "Searches on this record server return 20 results per page, in no time order. Add &_sort=-date&_count=5 to a search to get the most recent results first." The results list is headed "Potassium results in this patient's record, most recent first (code K)" and has one row for each result, with the time the specimen was taken, the time the result was reported, the value with its unit, and the status.

With the two sentences added, the pass rate was 30 percentage points higher than with nothing added (95% CI 10 to 45). In more runs the agent also stopped after the search under 2823-3 (57 against 25), and with one run per instance that increase cannot be attributed to the two sentences.

The results list has eight rows, or more when more than eight results share the most recent time, so that it always contains every final result at the patient's most recent effective time. That is the condition under which it was proved in Lean that a subset of the results gives the same outcome as the full history (What is specified, and what is proved?). The number eight was chosen for this study, and the proved condition requires only the results at the most recent time. With the results list in its context the model ordered the required dose and wrote both orders on 195 of 200 instances, 58 percentage points more than with nothing added (95% CI 41 to 72).

Of the 5 runs that failed with the results list, in 2 the agent wrote the test order's priority as `routine` and in 2 it left the route out of the medication order. In the remaining 1 the agent submitted no order on the instance whose most recent result is recorded as 0.0 with no unit, and its final answer said that this was not a valid result. Both graders fail that run (What does a pass mean clinically?).

The two sentences state how to sort a search, and the results list contains the patient's most recent results. The model is the same in every condition. When the results are in its context the model computes the required dose and writes both orders on 195 of 200 instances. The agent still sent its own search in most of those runs, so it cannot be concluded that the model's orders came from the results list.

11 Six runs, step by step

The six runs below are runs of the model from the study. Each was selected by a fixed rule, given with the run, that uses only recorded properties of the run and its instance. The rules were written to find one example of each of six outcomes, so the six runs are illustrations and are not a random sample. Each panel contains what the task's rule requires, the patient's results, what each server response contained, the orders the agent submitted, and the two graders' verdicts.

The agent's own messages are given exactly as sent. The server's responses are summarised, because each contains the full FHIR resource of every result.

The potassium values are from the benchmark's released, de-identified records. The patient identifiers in the messages are the ones the mock server used for the derived instances, and are not the benchmark's identifiers. Times are shown as days before the patient's most recent result.

11.1 Run 1. Most recent result on page 2

Required by the task's rule. The patient's most recent result is 2.8 mmol/L, which is below 3.5. Two orders are required: 70 mEq of the named product by mouth, and a potassium test at 08:00 on 14 November.



28 potassium results. Vertical axis: mmol/L, with a dashed line at 3.5. Horizontal axis: days before the patient's most recent result.

1. **The agent searched.** `GET Observation?code=2823-3&patient=S8004797` The server returned no results.
2. **The agent searched.** `GET Observation?code=K&patient=S8004797` The server returned 20 of the 28 results and a link to the next page. The most recent of the 20 was 3.9 mmol/L, taken 71 minutes earlier than the patient's most recent result, which was not among them.
3. **The agent finished with a list containing this text.** "Patient S8004797's most recent serum potassium level is 3.9 mmol/L, collected on 2023-11-13 at 09:04. This is not low (goal 3.5 mmol/L), so no replacement potassium order was placed and no paired next-day morning serum potassium level was ordered."

Relaxed grader *fail.* No order was submitted.

Released grader *fail.* It fails because the two orders are missing; the final answer is a list of sentences, and the grader requires an empty list or the potassium value.

What happened The first page contained 20 of the 28 results, in order of record id. The most recent of those 20 was 3.9 mmol/L, which is not below 3.5, and the agent submitted no order. The patient's most recent result was on the second page, which the agent did not request.

Why this run is shown It is the smallest potassium history of more than 20 results on which the run with the default sort order read only the first page and submitted no order, and the run on the same instance with the most recent results first passed.

The agent's messages, exactly as sent, are on the web version.

11.2 Run 2. Same instance, most recent first

Required by the task's rule. The patient's most recent result is 2.8 mmol/L, which is below 3.5. Two orders are required: 70 mEq of the named product by mouth, and a potassium test at 08:00 on 14 November.



28 potassium results. Vertical axis: mmol/L, with a dashed line at 3.5. Horizontal axis: days before the patient's most recent result.

1. **The agent searched.** `GET Observation?code=2823-3&patient=S8004797` The server returned no results.
2. **The agent searched.** `GET Observation?code=K&patient=S8004797` The server returned 20 of the 28 results and a link to the next page. The patient's most recent result was among them.
3. **The agent submitted an order.** A medication order for the product with NDC 40032-917-01: **70 mEq**, route written as `{"text": "Oral"}` .
4. **The agent submitted an order.** An order for a test with LOINC code 2823-3 at **2023-11-14T08:00:00+00:00**, priority stat.
5. **The agent finished with a list containing this text.** "Patient S8004797's most recent potassium level was 2.8 mmol/L on 2023-11-13T10:15:00+00:00, which is low (goal 3.5 mmol/L). Calculated oral potassium replacement: deficit 0.7 mmol/L; 0.7 / 0.1 = 7 increments; 7 × 10 mEq = 70 mEq. Ordered potassium oral replacement, NDC 40032-917-01, 70 mEq orally, status active, intent order. Paired order placed for morning serum potassium level (LOINC 2823-3) to be completed 2023-11-14T08:00:00+00:00."

Relaxed grader *pass.* Both orders, with the required dose and every required field.

Released grader *fail.* It fails because the route is the declared object, and the grader requires a plain string; the final answer is a list of sentences, and the grader requires an empty list or the potassium value.

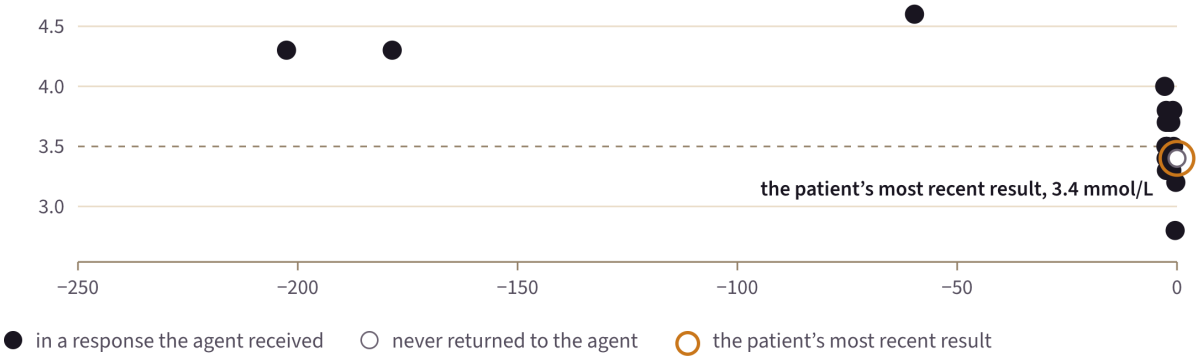
What happened The potassium history and the prompt are the same as in run 1. The mock server listed the most recent results first, the patient's most recent result was first on the page, and the agent wrote the two required orders.

Why this run is shown It is the same instance as the previous run, under the other condition.

The agent's messages, exactly as sent, are on the web version.

11.3 Run 3. Dose from an older result

Required by the task's rule. The patient's most recent result is 3.4 mmol/L, which is below 3.5. Two orders are required: 10 mEq of the named product by mouth, and a potassium test at 08:00 on 14 November.



21 potassium results. Vertical axis: mmol/L, with a dashed line at 3.5. Horizontal axis: days before the patient's most recent result.

1. **The agent searched.** `GET Observation?code=2823-3&patient=S8003610` The server returned no results.
2. **The agent searched.** `GET Observation?code=K&patient=S8003610` The server returned 20 of the 21 results and a link to the next page. The most recent of the 20 was 3.2 mmol/L, taken 6 hours earlier than the patient's most recent result, which was not among them.
3. **The agent submitted an order.** A medication order for the product with NDC 40032-917-01: **30 mEq**, route written as `{"text": "oral"}` .
4. **The agent submitted an order.** An order for a test with LOINC code 2823-3 at **2023-11-14T08:00:00+00:00**, priority stat.
5. **The agent finished with a list containing this text.** "Patient S8003610's most recent potassium level was 3.2 mmol/L on 2023-11-13T03:00:00+00:00, which is low (goal 3.5). Deficit 0.3 mmol/L → ordered 30 mEq oral potassium replacement (NDC 40032-917-01). Also ordered a morning serum potassium level (LOINC 2823-3) for 2023-11-14T08:00:00+00:00."

Relaxed grader *fail.* The dose is not the required dose.

Released grader *fail.* It fails because the route is the declared object, and the grader requires a plain string; the dose is not the required dose; the final answer is a list of sentences, and the grader requires an empty list or the potassium value.

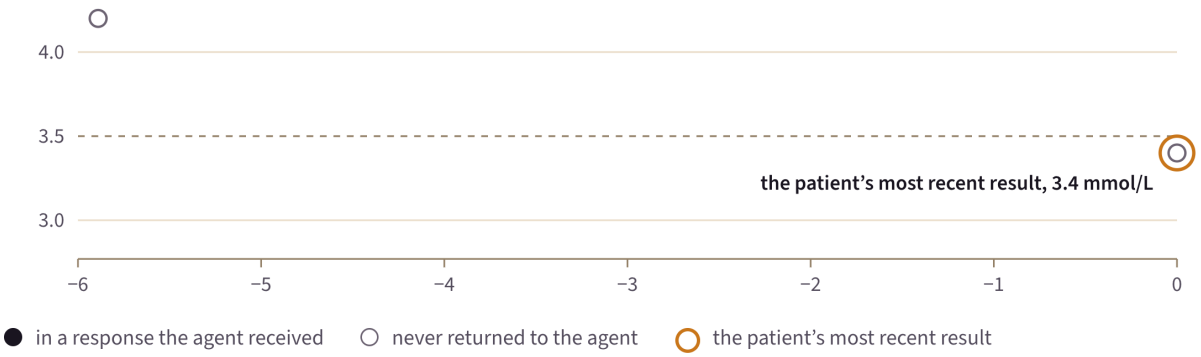
What happened The most recent result on the first page was 3.2 mmol/L, and the agent ordered 30 mEq, which is the rule applied to that value. The patient's most recent result, 3.4 mmol/L, was on the second page, and the rule applied to it gives 10 mEq.

Why this run is shown It is the smallest potassium history on which the run ordered a dose that equals the rule applied to the most recent result on the first page, which was not the patient's most recent result.

The agent's messages, exactly as sent, are on the web version.

11.4 Run 4. Searched only under 2823-3

Required by the task's rule. The patient's most recent result is 3.4 mmol/L, which is below 3.5. Two orders are required: 10 mEq of the named product by mouth, and a potassium test at 08:00 on 14 November.



2 potassium results. Vertical axis: mmol/L, with a dashed line at 3.5. Horizontal axis: days before the patient's most recent result.

1. **The agent searched.** `GET Observation?code=2823-3&patient=S8001878` The server returned no results.
2. **The agent finished with a list containing this text.** "No serum potassium level (LOINC 2823-3) was found for patient S8001878, so the most recent potassium level could not be determined. Consequently, no potassium replacement order was placed and no follow-up morning serum potassium level was scheduled."

Relaxed grader *fail.* No order was submitted.

Released grader *fail.* It fails because the two orders are missing; the final answer is a list of sentences, and the grader requires an empty list or the potassium value.

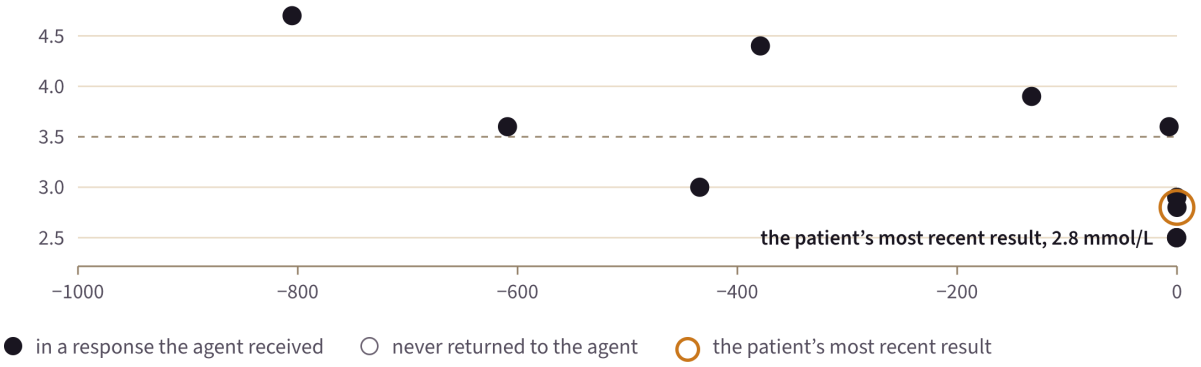
What happened The task context gives 2823-3 as the LOINC code for the repeat test. The record server stores potassium results under the code K. The search under 2823-3 returned an empty page, and the agent reported that the patient had no potassium result.

Why this run is shown It is the smallest potassium history on which the run searched only under the code 2823-3.

The agent's messages, exactly as sent, are on the web version.

11.5 Run 5. A history of 12 results

Required by the task's rule. The patient's most recent result is 2.8 mmol/L, which is below 3.5. Two orders are required: 70 mEq of the named product by mouth, and a potassium test at 08:00 on 14 November.



12 potassium results. Vertical axis: mmol/L, with a dashed line at 3.5. Horizontal axis: days before the patient's most recent result.

1. **The agent searched.** `GET Observation?code=2823-3&patient=S8000056` The server returned no results.

2. **The agent searched.** `GET Observation?code=K&patient=S8000056` The server returned 12 of the 12 results. The patient's most recent result was among them.
3. **The agent submitted an order.** A medication order for the product with NDC 40032-917-01: **70 mEq**, route written as `{"text": "oral"}` .
4. **The agent submitted an order.** An order for a test with LOINC code 2823-3 at **2023-11-14T08:00:00+00:00**, priority stat.
5. **The agent finished with a list containing this text.** "Patient S8000056's most recent serum potassium was 2.8 mmol/L on 2023-11-13T09:31:00+00:00, which is low (goal 3.5 mmol/L). Calculated oral potassium replacement: 70 mEq (0.7 mmol/L below goal × 10 mEq per 0.1 mmol/L). A MedicationRequest for replacement potassium 70 mEq oral was placed. A paired ServiceRequest for morning serum potassium level on 2023-11-14 at 08:00 was also placed."

Relaxed grader *pass.* Both orders, with the required dose and every required field.

Released grader *fail.* It fails because the route is the declared object, and the grader requires a plain string; the final answer is a list of sentences, and the grader requires an empty list or the potassium value.

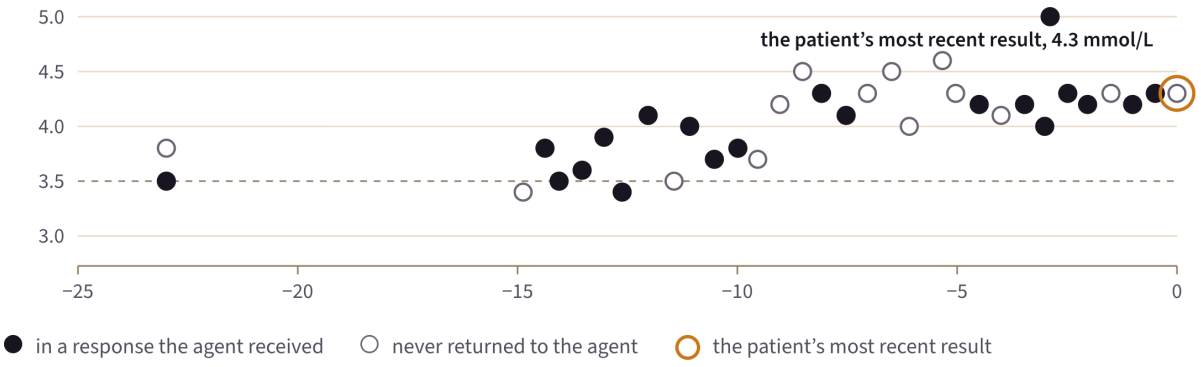
What happened The server returned all 12 results on one page, and the agent's medication order has the dose that the rule gives for the most recent of them.

Why this run is shown It is the passing run whose potassium history is closest to 12 results, a size chosen to show a history that fits on one page.

The agent's messages, exactly as sent, are on the web version.

11.6 Run 6. A no-action instance

Required by the task's rule. The patient's most recent result is 4.3 mmol/L, which is not below 3.5. No order is required.



34 potassium results. Vertical axis: mmol/L, with a dashed line at 3.5. Horizontal axis: days before the patient's most recent result.

1. **The agent searched.** `GET Observation?code=2823-3&patient=S8000118` The server returned no results.
2. **The agent searched.** `GET Observation?code=K&patient=S8000118` The server returned 20 of the 34 results and a link to the next page. The most recent of the 20 was 4.3 mmol/L, taken 12 hours earlier than the patient's most recent result, which was not among them.
3. **The agent finished with a list containing this text.** "Patient S8000118's most recent potassium level was 4.3 mmol/L (effective 2023-11-12T21:09:00+00:00), which is within normal range and above the 3.5 mmol/L goal. No potassium replacement order was placed, and therefore no paired morning serum potassium level was ordered."

Relaxed grader *pass.* No order was required and none was submitted.
Released grader *fail.* It fails because the final answer is a list of sentences, and the grader requires an empty list or the potassium value.
What happened The agent read the first page, where the most recent result was 4.3 mmol/L, and submitted no order, which is what the grader requires. The patient's most recent result was on a later page, and it is also 4.3 mmol/L.
Why this run is shown It is the smallest potassium history of more than 20 results on a no-action instance where the run submitted no order and no response contained the patient's most recent result.

The agent's messages, exactly as sent, are on the web version.

12 Does a model-written program read past the first page?

In the runs above the model writes the agent's message in each round of each instance. The task is the same computation for every patient, so the model can instead be asked to write it once as code, and the code can be run on every instance. Shi and coauthors asked language models to write code that operates on patient records,[11] and the authors of FHIR-AgentBench compare code generation with reasoning in natural language.[3] Here it was tested whether the code the model writes retrieves the patient's most recent result. The counts in this section are of functions, and the counts above are of runs.

DeepSeek V4.1 Flash was asked 20 times, at temperature 1, for one Python function that does the whole task with the agent's tools: the function takes as arguments a function for sending GET requests, the server's address and the patient identifier, and returns the orders and the final answer. The prompt contained the benchmark's tool definitions and the task's instruction and context. It contained nothing about paging, sorting or the limit of 8 rounds.

Each of the 20 functions was then revised once. Each function was run on a synthetic potassium history of 30 results, served by the mock server, in which the most recent result is on the second page. The model was sent its function, the requests the function had made, the responses and what the function returned, and was asked to revise the function if it needed revising. The model was not told the expected orders, or whether the function's orders were the expected ones.

The 20 functions and their 20 revisions were run on all 7,590 derived instances and on the 30 released instances, against the mock server with its default sort order, and scored by the relaxed grader. Each function was run with a limit of 200 GET requests and 60 seconds per instance, which no function reached, and again within the benchmark's limit of 8 rounds. The table contains the action-required and no-action instances. The tied instances are left out, because the instruction does not state what is required on them.

Table 9: Functions, of 20, that pass on every instance of a group

	All 629 action-required instances	All 6,705 no-action instances	All 30 released instances	Median instances passed, of 629
The 20 functions written from the task text and the tool definitions	**0**	2	16	299
The same functions after one revision	**14**	16	18	629
The revised functions, within 8 rounds	**1**	2	3	325.5

Before revision. All 20 functions read only the first page of their search. Of the 20, 14 return the same outcome as the first-page baseline, including the dose, on every one of the 7,590 derived instances, and 17 do on all 629 action-required instances. Two search only under 2823-3 and submit no order on any instance. When asked for a program, the model wrote code that reads only the first page, as in most of the agent's runs.

After revision. Of the 20 functions, 17 follow the link to the next page, and one of those also requests a larger page with `_count`. One sets `_sort`. One still reads only the first page, and one sends only a search with no code. The mock server, which contains only potassium results, returned the first page of those. The released server would also return the patient's other observations to such a search, which was not run, so these runs do not reproduce the released server. That function passes on no action-required instance. With up to 200 GET requests allowed for each instance, 14 pass on all 629 action-required instances, and the same 14 also pass on all 6,705 no-action instances. Within 8 rounds, one does: the function that sets `_sort`. Two orders and FINISH leave five rounds for searches, five pages contain at most 100 results, and 255 of the 629 instances have a history of more than 100. On each of four released instances with long histories, between 5 and 16 of the 20 revised functions reached the limit of 8 rounds.

These counts have three limits. First, the limit of 8 rounds was not in the prompt, so these counts measure what the revision produced under a limit the model was not told about. They do not measure whether the model could write a function for that limit. Second, in the revision step the model received its own code, the requests and responses, and a second attempt, on a history in which the most recent result is on the second page, so the increase in passing functions cannot be attributed to any one of these three. Third, 2 of the first 20 functions return the first-page baseline's outcomes but write the patient reference in a form the grader does not accept, so they pass on no action-required instance.

The released grader passes no output of any function on any instance, for the same two reasons as the agent's runs: the route written as an object, and the form of the final answer.

13 What is specified, and what is proved?

The outcome required on each derived instance is defined by a formal specification of the instruction and computed by a reference implementation, both written in Lean.[12] The benchmark's authors call their grading code a reference solution; the reference implementation here is a separate program written for this study. In earlier formal work on MedAgentBench, Mondl and coauthors translate agent instructions and policy documents into machine-checkable safety policies and measure how much of a synthetic policy the translated policies cover.[13] Here, the required outcome of one task is specified.

For each of the four kinds of program used in this document, the table contains what it does and the evidence that it does so correctly.

Program	What it does	Evidence about it
The reference implementation, in Lean	From a list of results and a clock time, computes the outcome the specification requires: no order, or the two orders with a dose.	Proved in Lean to meet the specification for every list of results.
The scripted baselines, in Python	Search the record server and submit orders through the benchmark's protocol.	Tested: run on every instance and scored by the released grader.
The model-written functions, in Python	Search the record server and return orders and a final answer, as code written by the model.	Tested: run on every derived instance by a separate harness and scored by the relaxed grader. Nothing is proved about them.
The released grader, in Python	Returns pass or fail for a run.	Used as released. A Lean description of how it selects a result and a dose is proved to agree with the specification under the conditions given below.

13.1 Choices made in the specification

On four points the text of the instruction and context does not determine the outcome. For each point the table contains the choice made in the specification, what the released grader's code does, and the effect on the instances used here.

Open point	The specification	The released grader	Effect on the instances used here
Which results count	Results with the status `final` and an effective time at or before the task's clock.	Every result, whatever its status or time.	None. All 7,702 released results are final, and none is dated after the clock. FHIR defines `amended` and `corrected` as revisions of a final result,[14] so results with those statuses would require a rule that is not yet in the specification.
Tied results with different values	No order.	The first of the tied results that the server lists.	Such a tie occurs on 256 derived instances. On 227 of them every tied value is 3.5 or above, so the specification and the grader both require no order. On the other 29 the dosing rule gives different outcomes for different tied values: on 12 the tied values are on both sides of 3.5, and on 17 every tied value is below 3.5 and the doses differ. The grader requires the two orders on 24 of the 29, the instances on which the first tied result that the server lists is below 3.5. The specification requires no order on any of the 29.

Open point	The specification	The released grader	Effect on the instances used here
A result with no unit	Read as mmol/L.	The value is used as it is, which gives the same number.	No unit is recorded for 6 results. One is recorded as 0.0, and on the instance where it is the most recent result the dosing rule gives 350 mEq.
"the next day at 8am"	08:00:00 on the next calendar day, on the task's clock.	Any time containing the text `2023-11-14T08:`.	In each of the model's 1,109 runs that the relaxed grader passes, in the two sort-order conditions and the two conditions with added information, the test order's time also meets the specification's reading. The task's clock is in UTC, and the records contain no ward time zone.

None of these choices was made by a clinician. Two tied values at one effective time may be two specimens, two methods or a duplicate, and the released records do not state which. A rule for tied results would require that information, or a convention set by the benchmark's authors. Under the released grader the null agent passes 232 of the 256 tied instances, which are the instances on which the grader requires no order, and the first-page baseline passes 242.

On the 29 tied instances where the dosing rule gives different outcomes for different tied values, the results list was added to the task context, so that all the tied results were in the model's context. Of the 29, the agent submitted no order on 4. It ordered a dose from the lowest tied value on 17 and from the highest on 8, and on none did its final answer state that the tied values differ. The specification's rule was then added in two sentences: "Rules for reading these results: a result counts only if its status is final. If the counted results taken at the most recent time have different values, the task cannot be decided from the record: place no order and say so when you finish." With them added, the agent submitted no order on 29 of 29, and its final answer stated that the tied values differ on 29. The two sentences request both actions, so in these runs the agent's actions matched the rule when the rule was in the task context. That is not evidence that the rule is the one a clinician would choose. Claude read each of the 58 final answers to confirm these counts.

13.2 What was verified in Lean

The specification, the reference implementation and the proofs are in one file, and Lean's kernel checked the proof of every theorem below. Each theorem is stated for lists of results reduced to three fields: an effective time, a value in tenths of a mmol/L, and whether the status is final.

- For every list and every clock time, the reference implementation returns an outcome that satisfies the specification, and exactly one outcome satisfies it. The expected outcome on each of the 7,590 derived instances is therefore determined by the specification. `run_meets_spec spec_unique`
- The outcome does not depend on the order in which the results are listed, so the server's sort order cannot change the required outcome. `run_order_free`
- A shorter list gives the same outcome as the patient's full history if it contains every final result that has the most recent effective time at or before the clock, and nothing that is not in the history. A search that returns every final result at the most recent time therefore returns enough results to determine the required outcome, and no other result is needed.

The results list used above was built to meet the condition, and the first-page baseline fails only on instances where the first page does not meet it. `run_sufficient_view`
- The theorem is false without the condition: Lean's kernel checked an example in which a list missing one of two tied results has a different outcome. `truncated_view_can_differ`
- A Lean description of how the released grader selects a result and a dose gives the same outcome as the specification for any history in which every result is final, none is dated after the clock, and no two results with the same effective time differ. `grader_agrees` Some histories used here have such a tie at an earlier time, which the theorem does not cover, so the agreement was also tested by running the programs: the all-results baseline orders the specification's dose on all 629 action-required instances and submits no order on all 6,705 no-action instances, and the released grader passes it on every one of them.

The proofs do not cover the code that reads a stored record into those three fields, the mock server, or the scoring of a run. No theorem states whether the specification is the right clinical reading of the instruction.

The specification in Lean

```
/-- One potassium result as the record holds it. -/
structure Obs where
  /-- When the specimen was taken (`effectiveDateTime`). -/
  time : Int
  /-- The result in tenths of a mmol/L. -/
  tenths : Int
  /-- Whether the record marks the result `final`. -/
  final : Bool
  deriving DecidableEq, Repr

/-- A result counts at decision time `now` when it is final and was taken at or before `now`. -/
def eligible (now : Int) (o : Obs) : Bool :=
  o.final && decide (o.time ≤ now)

/-- "Low" in the instruction: below 3.5 mmol/L. -/
def threshold : Int := 35

/-- "The next day at 8am": 08:00:00 on the calendar day after `now`, on the same UTC scale. -/
def nextMorning (now : Int) : Int :=
  (now / 86400 + 1) * 86400 + 8 * 3600

/-- What the task can come to. -/
inductive Outcome where
  /-- No eligible potassium result exists. Nothing is ordered. -/
  | noResult
  /-- The most recent eligible time carries results with different values.
      The instruction does not say which to use. Nothing is ordered. -/
  | conflict (time : Int)
  /-- The most recent result is 3.5 or above. Nothing is ordered. -/
  | notLow (time tenths : Int)
  /-- The most recent result is below 3.5: order `dose` mEq and a repeat potassium at `repeatAt`. -/
  | replace (time tenths dose repeatAt : Int)
  deriving DecidableEq, Repr

/-- `t` is the most recent eligible time. -/
def IsLatest (now : Int) (obs : List Obs) (t : Int) : Prop :=
  (∃ o ∈ obs, eligible now o = true ∧ o.time = t) ∧
    ∀ o ∈ obs, eligible now o = true → o.time ≤ t

/-- Every eligible result taken at `t` has the value `v`. -/
def AgreeAt (now : Int) (obs : List Obs) (t v : Int) : Prop :=
  ∀ o ∈ obs, eligible now o = true → o.time = t → o.tenths = v
```

```
/-- The statement of the task: which outcome the instruction requires for a given record.
    It is written with "for all" and "there exists" over the results, with no algorithm in it. -/
def Spec (now : Int) (obs : List Obs) : Outcome → Prop
  | .noResult => ∀ o ∈ obs, eligible now o = false
  | .conflict t =>
      IsLatest now obs t ∧
        ∃ a ∈ obs, ∃ b ∈ obs, eligible now a = true ∧ eligible now b = true ∧
          a.time = t ∧ b.time = t ∧ a.tenths ≠ b.tenths
  | .notLow t v => IsLatest now obs t ∧ AgreeAt now obs t v ∧ threshold ≤ v
  | .replace t v dose repeatAt =>
      IsLatest now obs t ∧ AgreeAt now obs t v ∧ v < threshold ∧
        dose = 10 * (threshold - v) ∧ repeatAt = nextMorning now
```

14 What does a pass mean clinically?

The sections above are about whether a run can pass on the released instances when the agent has not received the patient's most recent result. This section is about what the benchmark's rule requires an agent to order. Below, the required doses are compared with one product label. This comparison is not a clinical review, and no clinician has assessed these orders.

The rule gives 10 mEq for each 0.1 mmol/L below 3.5, written into one dose field. A most recent result of 3.1 gives 40 mEq, and a result of 2.8 gives 70 mEq. The one released instance that requires the orders has a most recent result of 3.1, so the released grader requires a 40 mEq order on it.

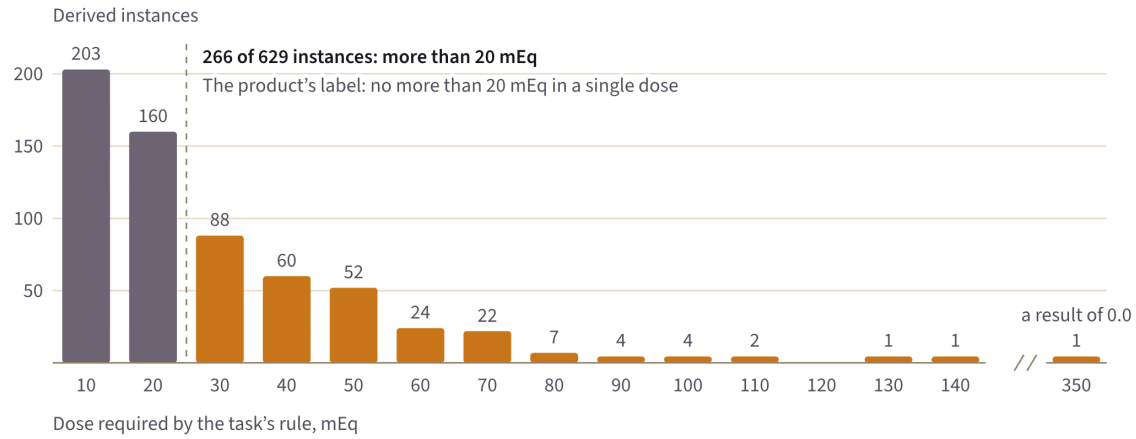


Figure 3: The dose the task's rule requires on each of the 629 action-required derived instances. The axis is broken between 140 and 350.

The product with the NDC in the task context is a potassium chloride extended-release tablet of 20 mEq. Its label gives 40 to 100 mEq per day or more for the treatment of potassium depletion, and states that a daily dose above 20 mEq should be divided so that no single dose exceeds 20 mEq.[15] FHIR defines the order's dose field as the amount given at one administration,[16] and the task text contains no schedule. As graded, the order on 266 of the 629 derived instances is therefore a single dose above 20 mEq of this product, and so is the 40 mEq order on the released instance. Whether the benchmark's authors intended a daily total is not stated in the task. This comparison is of the orders that the released grader passes; no patient's treatment is assessed.

The rule has no upper limit and no test of whether a value is plausible. The largest dose the rule gives here, 350 mEq, is for the result recorded as 0.0 with no unit, which is not a plausible measurement. On that instance, in the model's run with the most recent results first, the agent

ordered 350 mEq, and the relaxed grader passes that run. In the run with the results list in its context, the agent submitted no order and stated in its final answer that a potassium of 0.0 with no unit is not a valid result, and that run fails. Both graders require the 350 mEq order on this instance.

A pass on this task therefore means that the agent's orders match the benchmark's rule. It is not evidence that the order is appropriate for the patient, and on this instance the run whose final answer said the value is not a valid result is the one that fails. Blotske and coauthors, a group that includes clinical pharmacists, argue that the rule does not reflect the clinical complexity of potassium replacement, and they tested a model on vignettes written with fuller dosing guidance.[17]

15 Limitations

1. **One model, one setting.** The model's results are for DeepSeek V4.1 Flash at low reasoning effort and temperature 0. On the first 40 instances, the model passed 10 at high reasoning effort, against 12 at low effort. An agent that follows the link to the next page or sets `_sort` can receive the patient's most recent result. How often other models do so was not measured in this study. The results for the three scripted baselines involve no model.
2. **One run per instance.** When the run was repeated on 100 instances, the pass or fail result differed on 16. The counts are of these particular runs. The outcome on a single instance is not a stable property of the model.
3. **One version of the benchmark.** The task file, the tool definitions and the grader are those of the original benchmark at commit `9926011`. The corrected benchmark built by Mantravadi and coauthors could not be examined, because no code or data for it was found, so it was not tested whether a script that reads only the first page fails any of its instances.
4. **The derived instances are reconstructed, selected and clustered**, as set out under How were further instances derived? The confidence intervals come from resampling patients. They express the uncertainty in a rate that comes from which 59 patients are in the data, and they do not describe the rate at other hospitals or on other record systems.
5. **The mock server is not Stanford's server software.** On 2,200 test searches for the 100 real patients, it returned the same record ids in the same order, the same stated total and the same kinds of link as the released server in 2,191. In sorted searches, this comparison does not include the order among results that share one effective time; when that order is also compared, 2,157 are identical. All 1,600 searches without `_sort` were identical, and no run in the two sort-order conditions set `_sort`. The 9 differences are in sorted searches: in 8, results that share one effective time fall on different sides of a page boundary, and in 1 only the mock server states a total. The condition with the two added sentences and the model-written programs do use sorted searches. On the 30 released instances the model passed 30 of 30 through either server, and the agent sent the same requests in the same order on 14 of 30, against 16 of 30 for two runs on the released server itself.
6. **The relaxed grader is a choice made in this study.** It changes the form of the route and removes the final answer from scoring, which are the two undocumented requirements that every run of the model fails, and it keeps every other check. Removing the final answer means that a wrong statement in it is not counted. Under the released grader no run of the model passes.
7. **The proofs cover the specification and the reference implementation.** The code that reads records, serves searches and scores runs is not covered by the proofs.

8. **No clinician has reviewed this work.** The specification is one reading of the instruction. Under What does a pass mean clinically? the rule's doses are compared with one product label, and no patient's treatment is assessed.

16 Conclusion

On the 30 released instances of MedAgentBench's potassium task, the released grader gives the same score to a script that retrieves the patient's most recent result and to a script that reads only the first page of a search. From the benchmark's own records, 629 further instances that require the orders were derived. On these the first-page baseline passes 299 and the all-results baseline passes all 629.

With the server's default sort order, the agent requested only the first page in 542 of the model's 629 runs, and the model passed 255 under the relaxed grader. When the mock server listed the most recent results first, the model passed 521. On the first 200 instances, with the results list added to the task context, the model passed 195 of 200. In 369 of its 374 failed runs with the default sort order, the agent did not receive the patient's most recent result. When asked to write the whole task as a program from the same tool definitions, instruction and context, the model wrote functions that read only the first page.

Given these results, the benchmark's authors could make four changes to the task.

1. Instances could be added on which the patient's most recent result is on a later page and the dosing rule gives a different order for it than for the most recent result on the first page. From the existing records, 330 such instances that require the orders and 228 that require none can be derived.
2. The tool definition could state the order and page size of a search, or the task could state that discovering them is part of what is tested.
3. The grader could accept the route in the form the tool definition declares, and the task could state what the final answer must contain.
4. The task could state whether the dose is a single dose or a daily total, and what an agent should do with a result that is not plausible. These two are questions for the benchmark's clinical authors.

The model's numbers come from one model, one run per instance in each condition, and a grader with two checks relaxed, so they cannot be used to rank the model against others. No language model is involved in the comparison of the first-page and all-results baselines: the two scripts, which differ only in whether they retrieve the patient's most recent result, receive the same score on the 30 released instances and different scores on 330 of the 629 derived ones.

17 How this was made

I directed the work and reviewed this document. Claude wrote the Lean specification and proofs, the analysis scripts and the text of this document under my direction, and ran the experiments. Lean's kernel checked the proofs. The build script reads every count in the text from a results file, so no count is copied by hand. Reading the counts from files does not by itself establish that each sentence states correctly what its count counts.

The web page was then checked in further Claude sessions that had not written it. One read the web page from the top as a new reader and listed each term that was used before it was introduced. One compared each count and each description of code with the scripts, the results files, the benchmark's files and the released grader. One compared the wording with the papers

cited here. One reviewed the layout from screenshots at laptop and phone sizes. One then edited the text for vague wording and for labels that are not standard terms, and another compared the edited text with the earlier text and listed each sentence whose claim had changed. Every correction that these sessions listed was made. No person other than the author has checked this work, and no one outside the study has reviewed it.

The Lean files, the scripts, the three baselines, the model-written programs and the aggregate results are in the repository. A table in the repository has one row for each of the model's runs in six conditions (the released instances on the released server, the action-required instances with each sort order, the sample of no-action instances, and the two conditions with added information), with its searches, its verdicts and its row in the error analysis table, and no patient values. The repository does not contain the benchmark's patient records, Stanford's grader file or the full transcripts, which contain record text. The six runs shown step by step, the first of which is on the instance drawn in Figure 1, are the only extracts of patient records in this document. This document also contains two single values from the records: the 3.1 mmol/L of the released instance that requires the orders, and the result recorded as 0.0.

17.1 Every run of the model

The model was run 2,506 times in this study. 2,008 of the runs are in ten conditions reported above: the 30 released instances on the released server, run twice, and once through the mock server; the 629 action-required instances with each sort order; a sample of 120 no-action instances; the repeat on 100 instances; the two conditions with added information, on 200 instances each; and high reasoning effort on 40 instances. A further 58 are the runs on tied instances reported under What is specified, and what is proved?

The other 440 runs were made to test three questions that are not reported in any section above: what the agent orders on a sample of tied instances, what it orders when a result is marked entered-in-error, and whether adding the results list changes the outcomes on no-action instances. Their outcomes are in the table below, so that every run of the model is in this document. Runs on tied instances and on instances with an added result are scored against the specification's outcome, because the released grader counts a result whatever its status and takes the first of several tied results.

Condition	Runs	Outcome
80 of the 256 tied instances, with the default sort order and nothing added	80	The specification's outcome, which is no order, in 72. Of the other 8, 1 ended without a final answer and 7 submitted an order.
The same 80 tied instances, with the results list added	80	The specification's outcome in 64. Of the other 16, 6 ended without a final answer and 10 submitted an order.
40 action-required and 40 no-action instances, each with one newer result added that is marked entered-in-error and has a value on the other side of 3.5. Each was run with the results list, and again with the results list and the two-sentence rule	160	The specification's outcome, which does not count the added result, in 159. The released records contain no result with this status.
The sample of 120 no-action instances, with the results list added	120	Passed under the relaxed grader in 117. The other 3 ended without a final answer.

The benchmark files are at commit `9926011713` of its repository. The grader file is not in that repository; the benchmark's README contains a link to it. The copy used here has SHA-256 `67d19da263ac92c1…` and was run unmodified. The model was called through OpenRouter as `deepseek/deepseek-v4.1-flash-1017`.

References

1. Jiang Y, Black KC, Geng G, Park D, Zou J, Ng AY, Chen JH. MedAgentBench: a virtual EHR environment to benchmark medical LLM agents. *NEJM AI* 2025;2(9). doi:10.1056/AIdbp2500144. Preprint: arXiv:2501.14654. Code and tasks: github.com/stanfordml-group/MedAgentBench.
2. Mantravadi A, Rajgarhia H, Desikan P, Mukherji A. World feedback for clinical agents: diagnosing RL in FHIR environments. arXiv:2607.01470, 2026.
3. Lee G, Bach E, Yang E, Pollard T, Johnson A, Choi E, Jia Y, Lee JH. FHIR-AgentBench: benchmarking LLM agents for realistic interoperable EHR question answering. *Proc Mach Learn Res* 2026;297:1297–315. proceedings.mlr.press/v297/lee26a.html.
4. Liu R, Mohiuddin IQ, Schoeffler AJ, Renduchintala K, Nayak A, Vemu PL *et al.* PhysicianBench: evaluating LLM agents in real-world EHR environments. arXiv:2605.02240, 2026.
5. Liu Q, Zhang S, Qin G, Valanarasu JMJ, Huang X, Argaw P *et al.* HealthAgentBench: a unified benchmark suite of realistic agentic healthcare environments for challenging frontier AI agents. arXiv:2606.31179, 2026.
6. Bellibatlu RR, Wang Z, Zhang W. Do agent benchmarks do what they say? An executable-contract audit of tool-using agent environments. arXiv:2609.37315, 2026.
7. HL7. FHIR Release 4: Search. hl7.org/fhir/R4/search.html [Accessed 18 November 2026].
8. HAPI FHIR. JPA Server Starter, `AppProperties.java`, line 76 (`default_page_size = 20`), at commit `8db6228`. github.com/hapifhir/hapi-fhir-jpaserver-starter [Accessed 18 November 2026].
9. Anthropic. Demystifying evals for AI agents. 9 January 2026. anthropic.com/engineering/demystifying-evals-for-ai-agents [Accessed 18 November 2026].
10. HL7. FHIR Release 4: Observation, detailed descriptions. hl7.org/fhir/R4/observation-definitions.html [Accessed 18 November 2026].
11. Shi W, Xu R, Zhuang Y, Yu Y, Zhang J, Wu H *et al.* EHRAgent: code empowers large language models for few-shot complex tabular reasoning on electronic health records. arXiv:2401.07128, 2024.
12. de Moura L, Ullrich S. The Lean 4 theorem prover and programming language. In: *Automated Deduction (CADE 28)*. Springer, 2021:625–35. doi:10.1007/978-3-030-79876-5_37.
13. Mondl A, Maisel M, Brock JH. Autoformalization of agent instructions into policy-as-code. arXiv:2606.26649, 2026.
14. HL7. FHIR Release 4: value set ObservationStatus. hl7.org/fhir/R4/valueset-observation-status.html [Accessed 18 November 2026].
15. Novel Laboratories. Potassium chloride tablet, extended release: prescribing information (revised November 2016). DailyMed, US National Library of Medicine. dailymed.nlm.nih.gov, set id eb82527d [Accessed 18 November 2026].
16. HL7. FHIR Release 4: Dosage, detailed descriptions. hl7.org/fhir/R4/dosage-definitions.html [Accessed 18 November 2026].
17. Blotske K, Zhao X, Henry K, Murray B, Gao Y, Smith SE *et al.* Don't stop the heart: a performance analysis of large language models and potassium dosing. medRxiv [Preprint] 2026. doi:10.64898/2026.06.02.26354762.

Glossary

95% CI A 95% confidence interval from a cluster bootstrap: the patients are resampled at random with replacement 4,000 times, and all of a drawn patient's instances stay together. It expresses the uncertainty in the rate that comes from which patients are in the data. It does not include the variation between two runs on the same instance.

Action-required instance An instance on which the two orders are required, because the patient's most recent potassium result is below 3.5 mmol/L. On a no-action instance the most recent result is 3.5 or above and no order is required. The name no-action is taken from Mantravadi and coauthors.

All-results baseline The first-page baseline with one change: its search requests up to 5,000 results in one page, which is the request the released grader itself sends. It therefore receives every potassium result the patient has.

Default sort order The order in which the benchmark's server lists results when a request does not set _sort: ascending record id. The order of record ids is not the order of effective time.

Derived instance An instance built in this study from the benchmark's records: one of the benchmark's patients with only the potassium results that had been issued by a chosen earlier time, and with every time shifted by the same amount so that the chosen time equals the task's clock. The instruction and the context are unchanged.

Effective time The clinical time of a result, in the FHIR field effectiveDateTime. For a laboratory result it is usually the time the specimen was collected. In this document "most recent" means the latest effective time.

FHIR Fast Healthcare Interoperability Resources, the standard interface of the benchmark's record server. In FHIR a laboratory result is an Observation, a medication order is a MedicationRequest and an order for a test is a ServiceRequest.

Final The FHIR status of a result that is complete. The other statuses include preliminary, amended, corrected and entered-in-error.

First-page baseline A scripted baseline with no language model. It sends one search under the code K with no sorting and no page size, reads the one page that comes back, takes the result with the most recent effective time on that page, applies the dosing rule, and writes the orders in the form the released grader accepts. It never requests the next page.

Formal specification The task's instruction written in Lean as an exact condition. For any list of results and any clock time, exactly one outcome satisfies it. Where the text of the instruction does not determine the outcome, a choice was written into the specification, and each choice is listed in this document.

Instance One test case: one patient's record at one time, with the task's instruction and context. The benchmark releases 30 instances of the potassium task.

Issued time The time at which this version of a result was made available to clinicians, in FHIR's field issued.

Lean A programming language and proof checker. Lean accepts a theorem only when its kernel has checked every step of the proof.

LOINC A standard coding system for laboratory tests. 2823-3 is the LOINC code for potassium in serum or plasma.

mEq Milliequivalent, the unit in which potassium doses are written. For potassium, 1 mEq is 1 mmol.

Mock server A program written for this study. It stores one patient's potassium results in memory and responds to searches in the format of the benchmark's server. It was used for the derived instances, because the benchmark's server contains only each patient's complete record.

Most recent first The one change made in the second condition of this study: when a request does not set _sort, the mock server lists results in reverse chronological order of effective

time, the order that a FHIR client requests with _sort=-date. The prompt, the tool definitions and the records are unchanged.

NDC National Drug Code, the United States identifier for a medicinal product and pack size. 40032-917-01 is a bottle of 100 potassium chloride extended-release tablets of 20 mEq.

Null agent A baseline that sends no request and finishes at once with an empty answer. The name is from Mantravadi and coauthors.

Page One FHIR Bundle of a paged search: the results that one response to a search contains. On the benchmark's server a page contains 20 results unless the request sets _count, and the response includes a link to the next page.

Potassium history The potassium results that an instance contains for its patient. It is not the patient's whole record: no other part of the record is used.

Reference implementation A short Lean program that computes the outcome the specification requires. It is proved in Lean that its output meets the specification for every input.

Relaxed grader The released grader, with its code unchanged, applied to a copy of a run in which two things have been changed: an oral route written as the object that the tool definition declares is rewritten as the plain string, and the final answer is replaced by an empty list, which removes it from scoring. Every other check in the grader still applies. The model's runs in this document are scored with it, except the runs on tied instances and on instances with an added entered-in-error result, which are scored against the specification's outcome.

Released grader The Python function the benchmark's authors wrote to score the potassium task, distributed in the file refsol.py. It reads the transcript of a run and returns pass or fail.

Round One message from the agent and the response to it. The benchmark's harness, the program that passes messages between the agent and the record server, ends a run after 8 rounds.

Run One attempt on one instance through the benchmark's protocol: at most 8 rounds, each of them a GET request, a POST request or FINISH.

Tied results Two or three potassium results for one patient with the same effective time, which is the most recent in the patient's history. Instances with tied results whose values differ are reported separately in this document.

Tool definition The description of each permitted request that the benchmark gives the agent: its address, its parameters and their types.

Transcript The complete record of a run: every message the agent sent and every response it received.