

SherlockBench, Where Large Language Models Under-Perform Random Heuristics

Joseph Graham

2025-07-21

1 Abstract

In this paper I introduce SherlockBench, a framework for exploring how good LLMs are at pro-actively investigating a problem.

Through several experiments, we explore which parts of SherlockBench models find difficult. While analyzing results, we discover that the LLM investigations are worse than picking inputs randomly.

We then perform some experiments to improve the performance of models on the test, through both software assistance and fine-tuning.

We finally compare the different aspects of SherlockBench performance against Artificial Analysis, to show that we have isolated two additional distinct capabilities.

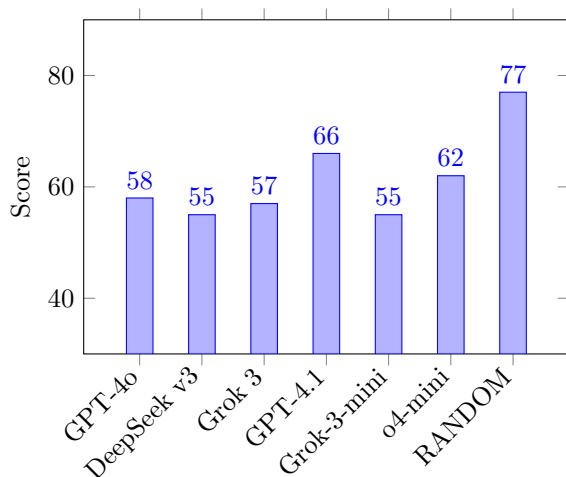


Figure 1: LLMs score 55-66% on investigation, vs. 77% for random heuristics. See section Isolating Investigation Performance.

2 Introduction

The effectiveness of AI in pattern matching and function simulation are well established[28, 29]. However I believe that pro-active investigation is an under-explored skill in AI models.

The most popular AI benchmarks such as MMLU[30], GPQA[31] and HLE[32] are essentially Q&A benchmarks.

There are several benchmarks for testing agentic behaviour such as AgentBench[33] and StableToolBench[34] but none explicitly test pro-active investigation.

To this end, we develop a novel test framework which requires an LLM to pro-actively investigate a mystery function to find out what it does.

Here is a brief overview of the structure of this paper:

- In Section 3, we describe the SherlockBench framework and experimental setup.
- In Section 4 we run the benchmark on the models and take an initial look at the results.
- In Section 5 we perform a series of experiments to isolate and measure investigation skill.
- In Section 6, we explore interventions to improve model performance, including software nudges and fine-tuning.

- Section 7 compares our findings to existing benchmarks.
- We conclude in Section 8 with discussion and future research.

2.1 Data and Code Availability

The full code and test logs are hosted externally due to size. Refer to the main text and bibliography for Zenodo links.

3 Experimental Setup

SherlockBench presents an LLM with a "mystery function" and asks it to investigate it using function calling (also known as "tool use").

This is done automatically by our benchmarking system, which consists of

- an API component [1] which holds the problem-sets, enforces the rules and judges the test results
- a client component [2] which runs the test loop and plugs different LLM providers into the SherlockBench API

The information the model is provided with is:

- the number and type of inputs the function takes
- the number of times it may test the function
- that its task is to investigate the function until it is confident it knows what it does

Based on that, it can call the function as many times as it wants up to a certain limit (20 for most problems).

The system then automatically confirms that the LLM has got the answer correct by giving it example inputs and asking it what the expected output will be.

3.1 Problem-Set

The problem-set used is here [3]. Here are some examples of the types of functions that are used:

3.1.1 "count vowels" :: str -> int

This returns the number of vowels in the input.

3.1.2 "add & subtract" :: (int, int, int) -> int

This adds the first two parameters and subtracts the last.

3.1.3 "interleave characters" :: (str, str) -> str

This interleaves the characters of two input strings.

3.1.4 "can form triangle" :: (int, int, int) -> bool

This returns true if the three integers can form the sides of a triangle.

3.2 2-Phase Mode and 3-Phase Mode

There are two ways of running SherlockBench, 2-phase mode and 3-phase mode.

In both cases, the first phase is the investigation-phase. This begins with the following prompt:

```
"role": "developer"
  You are a competent and alert chatbot.

  You are provided with a mystery function which you will "interrogate" to try to
  determine what it does. Use the provided tool to do this.

  Once you are confident you know what the function does, you will inform the user.
```

n.b. it is your job to pick inputs for the mystery function. Do not ask the user to provide you with parameters to test. Test the function pro-actively with the provided tool until you work out what the mystery function does.

"role": "user"

Hi. I have a mystery function and I want to find out what it does.

I would like you to test my function using the provided tool until you think you know what it does, then tell me.

You may test this function up-to {test_limit} times.

Once the LLM either uses all its tool calls or chooses not to call its tool, we move onto the next phase.

In 2-phase mode, the next and final phase is the verification phase. The LLM is asked several times to verify it understands what the function does, with prompts like this appended onto the conversation history from the investigation phase:

"role": "user"

To test your theory, please tell me what is the expected output from the function with this input:

{ 'a': 2, 'b': 4 }

You no-longer have access to the tool because I am testing if you have got it right.

Please respond in JSON with two keys: "thoughts" and "expected_output". expected_output should contain the output you expect from the function.

In the "2-phase" mode, investigating the function and deciding what it does are a single phase, with a single context window. In 3-phase mode, we split that apart by adding an extra step before the verification phase, called the decision phase.

How this works is that, after the investigation phase is completed, the conversation history is discarded, keeping only the tool call history (input/output pairs). The LLM is then given a single additional prompt to decide what the function does by looking at those. Then the verification phase proceeds as before, appending the verification prompts on-top of the conversation history from the decision phase.

The decision phase prompt looks like this:

"role": "developer"

You are a competent and alert chatbot. You will help to investigate a mystery function.

"role": "user"

I have a mystery function and I want you to figure out what it does.

Here are some examples of the function's input and output:

(1, 1) True
(2, 3) False
(5, 0) False
<redacted for brevity>

Based on these examples, please determine what the function does and summarize.

Here is a summary of the 2-phase and 3-phase test modes:

2-phase mode:

investigation phase

LLM uses tool calls to investigate the function and decide what it does

verification phase

test system gives the LLM a series of tests to confirm it correctly identified the function's behaviour

3-phase mode:

investigation phase

LLM uses tool calls to investigate the function

decision phase

given the input/output pairs from the tool calls from the investigation phase, decide what the function does

verification phase

test system gives the LLM a series of tests to confirm it correctly identified the function's behaviour

We provide a log which shows how an LLM (Deepseek v3 in this case) works through a problem: [11]

We provide test logs which show DeepSeek working through a problem in 2-phase mode [11] and 3-phase mode: [12].

We provide the raw JSON of the investigation phase of the 2-phase example: [13]

3.3 Featured models

For the purpose of this paper, we need a selection of different models to test. Not too many for cost reasons, but we want some diversity in type and capability.

So we pick four traditional LLMs, as well as two LRMs (reasoning models), shown in Table 1.

Table 1: AI Models tested in this Paper

type	provider	model name	API name	settings
LLM	OpenAI	GPT-4o	gpt-4o-2024-08-06	
	OpenAI	GPT-4.1	gpt-4.1-2025-04-14	
	xAI	Grok 3	grok-3	
	DeepSeek	Deepseek v3	deepseek-chat	
LRM	OpenAI	o4-mini	o4-mini-2025-04-16	"medium" effort
	xAI	Grok 3 mini	grok-3-mini	"high" effort

Note that henceforth we may abbreviate thusly in table headings:

- Deepseek v3 -> DS v3
- Grok 3 mini -> Gk3 mini

4 Initial Results

We run the test with 10 attempts per problem using 2-phase mode[14] and 3-phase mode[15], and present the results in Table 2 and Figure 2

Table 2: Performance of each model under 2-phase and 3-phase modes. Results are ordered by their performance on the 3-phase test.

Model	2-phase		3-phase	
	Average	pass@10	Average	pass@10
GPT-4o	12	42	14	42
deepseek v3	24	67	36	67
Grok 3	32	58	38	75
GPT-4.1	15	50	46	75
Grok 3 mini	20	75	48	75
o4-mini	70	83	65	83

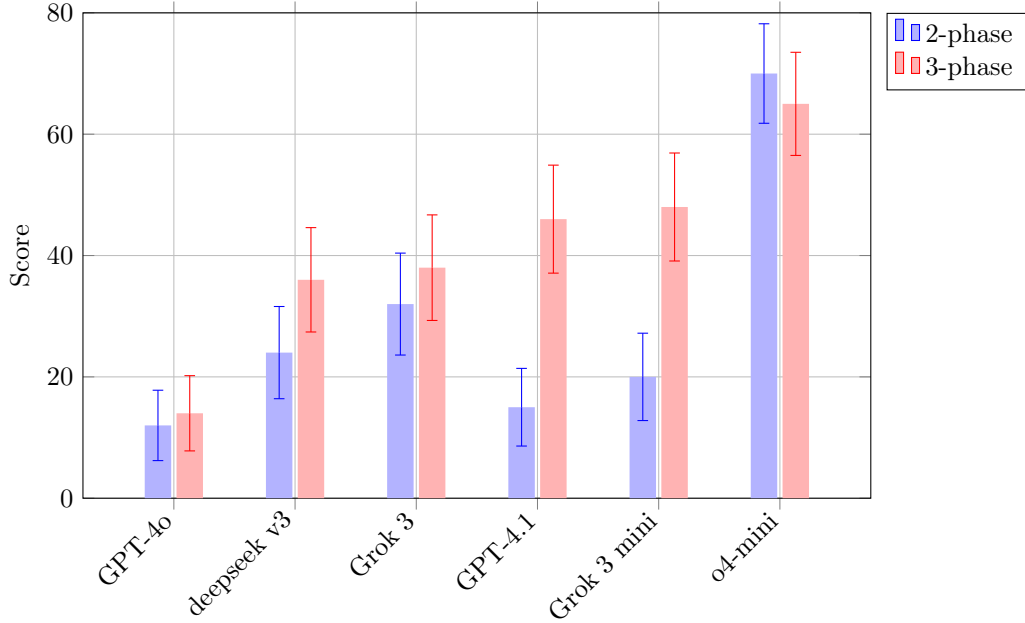


Figure 2: Average model performance on 2-phase and 3-phase mode. Results are ordered by their performance on the 3-phase test. Error bars: 95% CI.

As you can see, most models perform a lot better with the 3-phase method. This might be because the summarization of the tool calls helps to clean up their context window.

The 3-phase test will be used for all further experiments in this paper, because it generally makes models perform better and is more flexible for our experiments. We will also order the models by their performance on the 3-phase test.

4.1 Performance by Model and Problem

To understand how models perform at the 3-phase test, we show Figure 3.

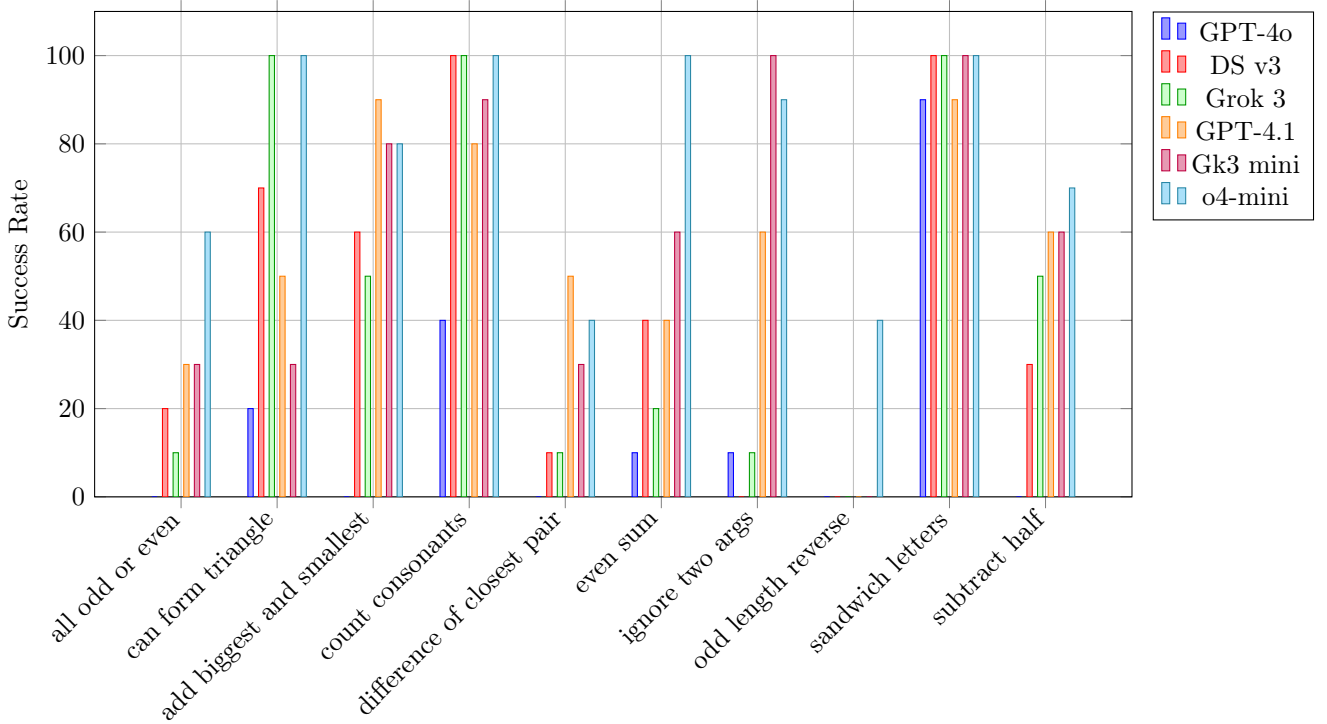


Figure 3: Success Rate of Each Model at Each Problem, as a percentage. Rendered from 10 samples per model+problem combination. Problems which no models can solve are omitted. Data Table at Appendix F.1.

As you can see, the models' performance at each problem is not very deterministic. There are a few which the models always get right or wrong, but most have a variable outcome.

4.2 Discussion

In order to understand exactly what SherlockBench is testing, we need to reflect on what skills the model needs to perform well at it.

At the high level, it needs to be able to:

investigate

try good combinations to test the model

pattern-match

detect patterns in the inputs and outputs

function simulation

reproduce the behaviour of the function accurately for a new set of inputs

It is well known that AI in general is good at pattern matching and function simulation, so the most interesting part of SherlockBench is the investigation phase.

What sort of skills might be needed to do well at the investigation phase? Here are some thoughts on what skills might be needed:

- pick good numbers to try
- form a hypothesis
- pick more numbers to test that hypothesis
- accept it's wrong and form a new hypothesis
- if running out of attempts, make a best guess

In this paper, we want to explore how good LLMs are at the investigation phase and why.

5 Characterizing Performance on Different Parts of the Test

5.1 Investigate Influence of Investigation Phase

Using the 3-phase test method, we can start to test the different phases of the test.

As a reminder; the three phases are:

investigation phase

test the function

decision phase

decide what the function does

verification phase

reproduce the function's behaviour with new data

It is assumed that success on the 3rd phase (verification) is mostly pre-determined by the 2nd phase (decision), so we group these phases together. So we are testing the investigation phase against the decision+verification phases.

For the rest of the paper we will abbreviate:

- Investigation Phase -> inv-phase
- Decision & Verification Phase -> d+v-phases

I introduce a new test mode which we will call inv-vs-dv. This is done using this code fork: [4].

The inv-vs-dv fork requires a problem-set of 1 single problem. For that problem it runs the inv-phase 10 times. Then for each of those, it completes the test a further 20 times each to get a success percentage for that investigation.

For each model we choose a problem which it **sometimes** gets right. This will allow us to analyse where the failures are coming from.

- GPT-4o with problem "count consonants"
- DeepSeek v3 with problem "even sum"

- Grok-3 with problem "add biggest and smallest"
- GPT-4.1 with problem "all odd or even"
- Grok-3-mini with problem "can form triangle"
- o4-mini with problem "difference of closest pair"

As a baseline we will also include a table produced by 10 "random" investigations using code fork "inv-vs-dv-random": [5]. The algorithm for generating the random investigations is at Appendix A.

5.1.1 Expected Results

Table 3		Table 4		Table 5		Table 6	
investigation	success %	investigation	success %	investigation	success %	investigation	success %
i1	100	i1	60	i1	50	i1	0
i3	100	i3	70	i2	50	i2	20
i2	0	i2	0	i3	40	i3	40
i4	0	i4	0	i4	60	i4	90

What do we expect the results to look like?

Imagine the data would look like Table 3, this would tell us:

- the inv-phase either succeeds or fails in a binary way
- the inv-phase is entirely pre-determining the over-all result

If it looks like Table 4, this would tell us:

- the inv-phase either succeeds or fails in a binary way
- the inv-phase partially pre-determines the over-all result, but the d+v-phases are contributing some failures also

If it looks like this Table 5 (binomial distribution), this would tell us:

- the inv-phase is not significantly pre-determining the over-all result
- most of the failures come from d+v-phases

If it looks like Table 6, this would tell us:

- the inv-phase is largely pre-determining the over-all result
- the investigation does not "succeed" or "fail" but rather has varying quality

5.1.2 Tests

We run the inv-vs-dv and inv-vs-dv-random tests with 10 attempts per problem [16, 17], and present our results:

1. GPT-4o with problem "count consonants"

Table 7: LLM investigation

investigation	success %
i1	5
i2	85
i3	20
i4	0
i5	5
i6	0
i7	0
i8	15
i9	70
i10	90

Table 8: Random Investigation

investigation	success %
i1	60
i2	90
i3	60
i4	25
i5	65
i6	20
i7	60
i8	85
i9	35
i10	55

Table 7 and Table 8 show the results for GPT-4o with problem "count consonants". The most immediate observation here is that random investigations perform better than LLM investigations. This was not expected.

Looking at the LLM investigation, we see a lot of low scores and high scores, with few intermediate scores. This tells us that:

- the inv-phase is largely pre-determining the over-all result
- the inv-phase does not simply succeed or fail, but has varying quality
- the d+v-phases are likely contributing some failures also

2. DeepSeek v3 with problem "even sum"

Table 9: LLM investigation

investigation	success %
i1	45
i2	80
i3	100
i4	80
i5	90
i6	0
i7	0
i8	75
i9	95
i10	75

Table 10: Random Investigation

investigation	success %
i1	85
i2	20
i3	95
i4	100
i5	100
i6	80
i7	80
i8	95
i9	95
i10	95

Table 9 and Table 10 show the results for DeepSeek v3 with problem "even sum". The narrative around this data is a similar story to GPT-4o.

3. Grok-3 with problem "add biggest and smallest"

Table 11: LLM investigation

investigation	success %
i1	20
i2	90
i3	85
i4	75
i5	85
i6	70
i7	10
i8	75
i9	40
i10	85

Table 12: Random Investigation

investigation	success %
i1	65
i2	60
i3	15
i4	55
i5	45
i6	35
i7	40
i8	45
i9	15
i10	40

Table 11 and Table 12 show the results for Grok-3 with problem "add biggest and smallest".

Like the previous, we have a range of low and high scores. However this is the first time that the LLM outperforms random investigations.

4. GPT-4.1 with problem "all odd or even"

Table 13: LLM investigation

investigation	success %
i1	10
i2	0
i3	5
i4	0
i5	0
i6	0
i7	40
i8	0
i9	10
i10	0

Table 14: Random Investigation

investigation	success %
i1	85
i2	65
i3	95
i4	80
i5	70
i6	5
i7	90
i8	90
i9	100
i10	0

Table 13 and Table 14 show the results for GPT-4.1 with problem "all odd or even".

This data again shows that the inv-phase is largely pre-determining the over-all result. However the random investigations outperform the LLM investigations particularly dramatically. It is worth digging into why that might be.

Looking at the logs for the failures, I can see that models often test lots of small and similar numbers which doesn't gather much useful information.

5. Grok-3-mini with problem "can form triangle"

Table 15: LLM investigation

investigation	success %
i1	0
i2	0
i3	0
i4	100
i5	100
i6	0
i7	65
i8	100
i9	15
i10	90

Table 16: Random Investigation

investigation	success %
i1	85
i2	80
i3	95
i4	100
i5	55
i6	90
i7	95
i8	100
i9	100
i10	80

Table 15 and Table 16 show the results for Grok-3-mini with problem "can form triangle".

Similar narrative as GPT-4o. However though the LLM investigation is closer to being a binary success or failure instead of having a variable quality.

6. o4-mini with problem "difference of closest pair"

Table 17: LLM investigation

investigation	success %
i1	65
i2	0
i3	45
i4	0
i5	0
i6	45
i7	100
i8	10
i9	0
i10	0

Table 18: Random Investigation

investigation	success %
i1	100
i2	100
i3	100
i4	100
i5	100
i6	100
i7	100
i8	100
i9	100
i10	100

Table 17 and Table 18 show the results for o4-mini with problem "difference of closest pair".

The narrative for this data is similar as for GPT-4o, though it's remarkable that random investigations always get 100%

5.1.3 Learnings

Over-all, we see that:

- the inv-phase is largely pre-determining the over-all result
- the investigation does not "succeed" or "fail" but rather has varying quality
- random investigations usually outperform LLM investigations

5.2 Isolating Investigation Performance

Our tests so-far have allowed us to see how the LLM performance is typically spread between the inv-phase vs d+v-phases.

However it would be good to see the performance of each model across the entire problem-set, not just on a specific problem chosen for each LLM.

I introduce a new test mode which we will call "standardized-dv", powered by this code fork[6]. And for comparison, we have a "standardized-dv-random" fork[7], which performs a random investigation (Appendix A).

This test mode isolates inv-phase performance from the d+v-phases performance by using the same model for d+v-phases every time. For each model we perform the inv-phase as normal, then use o4-mini for the d+v-phases.

We run the test with 10 attempts per problem [18, 19]. Figure 4 shows a summary of the results, with the original 3-phase results included for comparison:

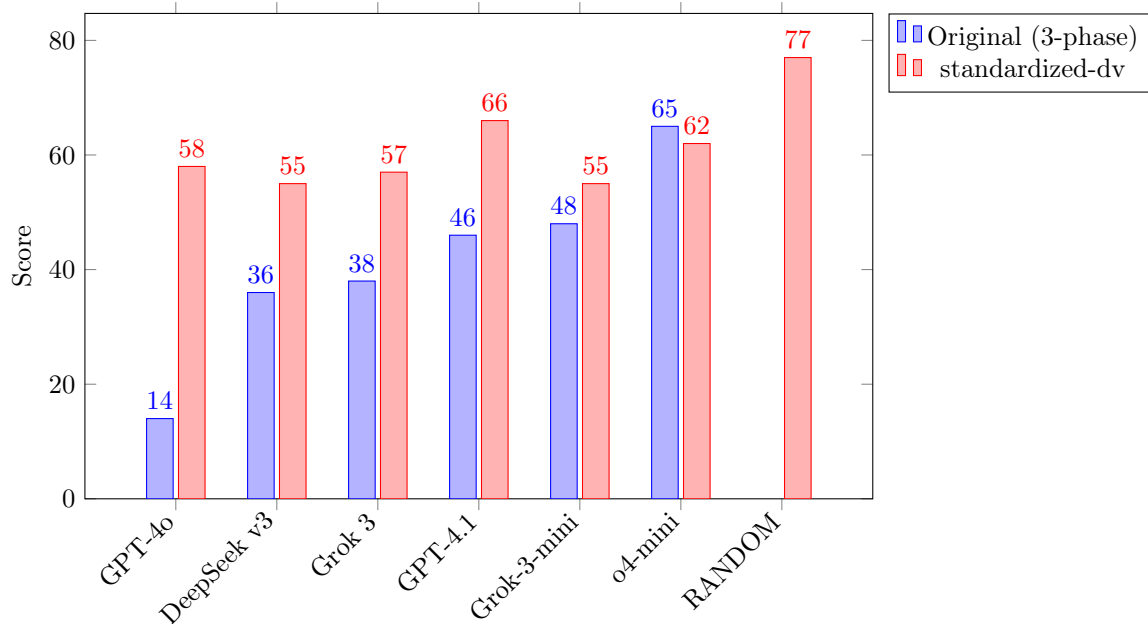


Figure 4: Investigation performance isolated vs original. Note that scores are expected to be higher as we are using our smartest model for the d+v-phases. The interesting part is the correlation or lack thereof. Data Table at Appendix F.2.

As you can see, models investigation performance does not correlate strongly to over-all performance.

Also none of them defeat the random investigation.

5.3 Isolating Decision+Verification Performance

Let's isolate d+v-phase performance. There are two ways we can do that. Either always use a random investigation, or record an LLM investigation and always use that, essentially making the test into a Q&A.

And so I introduce two new test modes:

- random-inv [8] runs the test using random investigations for all problems. Random algorithm at Appendix A
- standardized-inv [9] runs the test using pre-recorded investigations

The standardized investigation is assembled manually by taking examples from successful attempts at solving problems, except for two problems which none can solve. In these cases I just picked an arbitrary investigation from o4-mini. See the data-set here: [20].

We run the test with 10 attempts per problem [21, 22]. Figure 5 shows a summary of the results, with the original 3-phase results included for comparison.

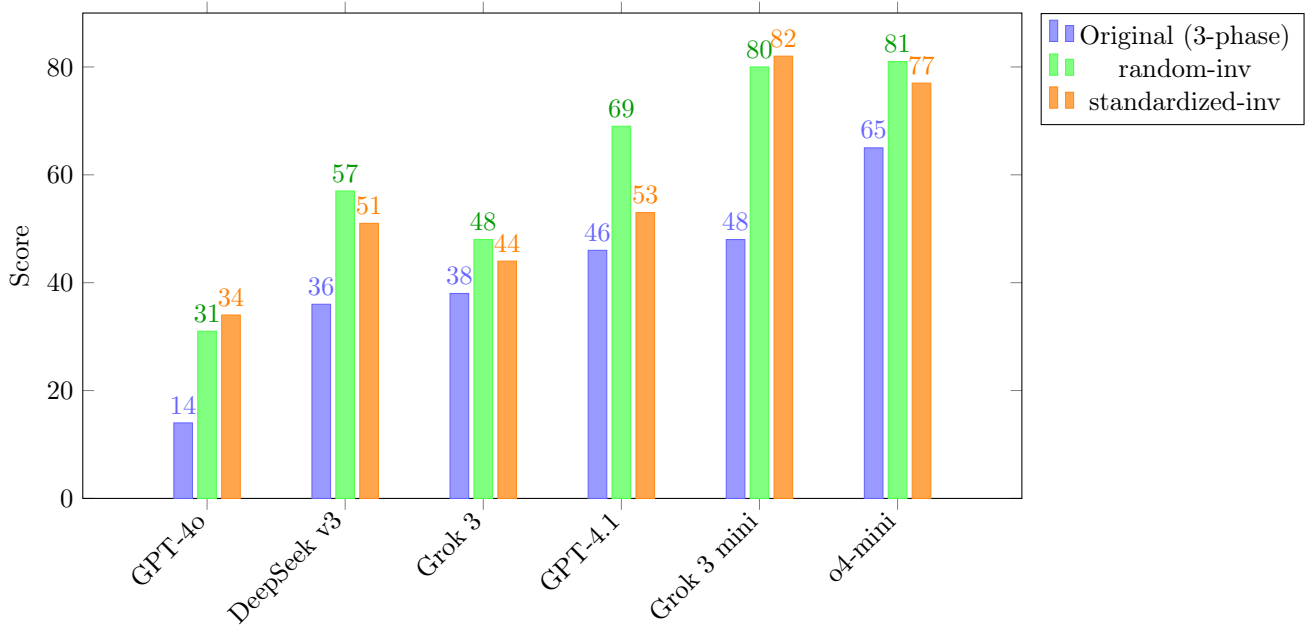


Figure 5: decision+verification performance isolated using two different methods. Data Table at Appendix F.3.

We can see that the performance on the full problem-set is always better with random investigations than with the "original" LLM investigation. This is unexpected and surprising.

Performance on the standardized investigation is also better than the original, which is expected, since these investigations were selected from successful attempts by a smart model. However it still usually under-performs the random investigation.

5.4 Analysing the Results

Let's put the data from the previous tests together:

- The basic 3-phase test
- Investigation performance isolated with standardized-dv
- Decision+Verification performance isolated with
 - random-inv
 - standardized-dv

In Figure 6 put the results from the previous tests together, by adding a line for the investigation performance.

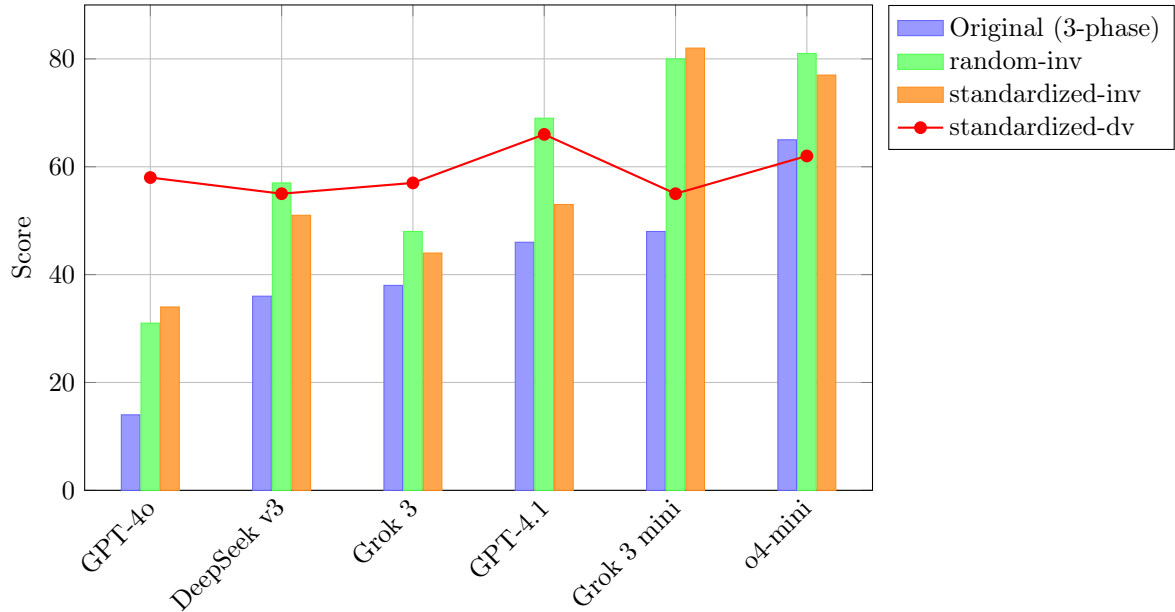


Figure 6: We show models’ d+v-phase performance (random-inv, standardized-inv) against it’s inv-phase performance (standardized-dv). Note the flat trend of the red line (investigation performance) compared to the general upward trend of the bars (decision+verification performance), highlighting the disconnect between these skills. Data Table at Appendix F.4

We can see from this that the higher performing models are getting their improved performance from being better at decision+verification, not from being better at investigation.

More broadly what we have learned so far is that the inv-phase is extremely important to the over-all result, and also that LLMs are very bad at it.

5.5 Discussion

Let us reflect on investigation performance. I notice two things which might explain why LLMs often under-perform random investigations:

- models often do not use all their tool calls, whereas the random investigation always uses the maximum allowed. See Appendix B.
- models often test lots of awkwardly small numbers like 0, 1, 2, whereas the random investigation are using larger numbers. See Appendix C

Therefore I would like to do some experiments to see if we can improve the LLM’s performance in these two areas.

6 Exploring Methods of Improving Performance

6.1 Software-level Intervention to Improve Investigation

Let’s see if we can assist the models’ performance at the software level by encouraging the LLM to try more combinations. I introduce code fork "software-assist" [10].

What this does is that, if the LLM stops testing its function with at-least 3 tests left, it receives an additional prompt like this:

You still have 5 messages left. Maybe try some more combinations to be confident.

We run the test with 10 attempts per problem [23]. Here is a summary of the results, with the original 3-phase test results as comparison:

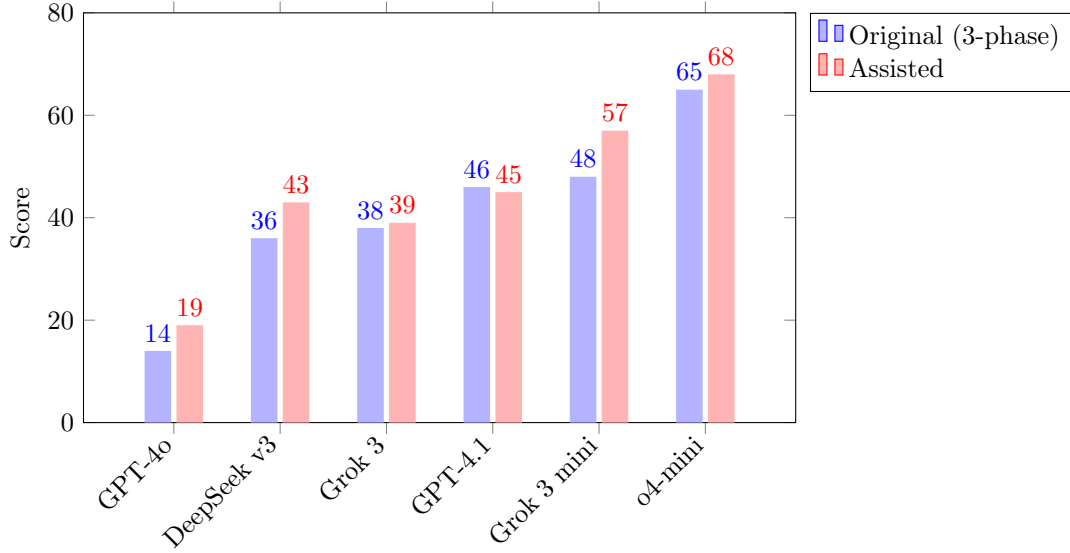


Figure 7: How good do models perform with software assistance. Data Table at Appendix F.5.

We can see that in all cases but one, the LLM has better performance like this.

GPT-4.1 is an anomaly. But looking at the test logs (Appendix B), we can see that it normally uses most of its tool calls even without software assistance, so makes sense for this model.

6.2 Fine-tuning

Can we use OpenAI’s fine-tuning API to tune a model to try bigger or randomer numbers?

To assemble a training data-set, I simply provide examples of the LLM calling its tool based on the prompt from the inv-phase. In these examples it calls the tool with random inputs generated as-per Appendix A.

Here is the manually-created dataset I created [24]. This is to be used with OpenAI’s fine-tuning API with supervised fine-tuning (SFT).

Using that data-set, I have made fine-tuned versions of GPT-4o and GPT-4.1. I will refer to these as GPT-4o-ft and GPT-4.1-ft.

We run the benchmark with the software-assist code fork with 10 attempts per problem [25]. Table 19 shows a summary of the results, with the non-fine-tuned models for comparison.

Table 19: fine-tuned models performance on software-assist code fork

model	percent
GPT-4o	19
GPT-4o-ft	12
GPT-4.1	45
GPT-4.1-ft	48

Essentially there is no improvement.

From looking at the log, I can see that the models are trying bigger numbers than usual, so the fine-tuning has worked. However I suspect the fine-tune has made the models worse at the d+v-phases. This is a common problem called Catastrophic Forgetting [35].

We run the benchmark with the "standardized-dv" code fork to isolate investigation performance from decision+verification performance [26]. Table 20 shows a summary of the results, with the non-fine-tuned models for comparison.

In this case there is a big improvement! It performs as well as a random investigation.

We run our "standardized-inv" code fork to better quantify how much the decision+verification performance degraded [27]. Table 21 shows a summary of the results, with the non-fine-tuned models for comparison:

We can clearly see that the fine-tuning has made the models better at investigation but worse at decision+verification.

Table 20: fine-tuned models performance on standardized-dv code fork

model	percent
GPT-4o	58
GPT-4o-ft	76
GPT-4.1	66
GPT-4.1-ft	80
RANDOM	77

Table 21: fine-tuned models performance on standardized-inv code fork

model	percent
GPT-4o	38
GPT-4o-ft	26
GPT-4.1	57
GPT-4.1-ft	52

6.3 Discussion

With software assistance and fine-tuning, we made the LLM try more combinations with bigger numbers. This made the models over-all stupider via Catastrophic Forgetting but it did improve investigation performance.

Testing more, bigger numbers could be described as a behaviour or strategy, not a raw measure of intelligence. So it would seem that LLMs do not have enough self-awareness to improve their own behaviours or strategies at test-time.

7 Comparison to Other Benchmarks

Is SherlockBench testing something different than other benchmarks?

In Table 22 we compare some of our tests to the Artificial Analysis Leaderboard[36]. Artificial Analysis is a meta-benchmark combining the results from several other benchmarks including MMLU-Pro, HLE and GPQA Diamond.

Table 22: Scores on the Artificial Analysis benchmark compared against SherlockBench

model	AA	2-phase	3-phase	standardized-dv	standardized-inv
GPT-4o	41	12	14	58	34
Grok 3	51	32	38	57	44
DeepSeek v3	53	24	36	55	51
GPT-4.1	53	15	46	66	53
Grok 3 mini	67	20	48	55	82
o4-mini	70	70	65	62	77
average	58.8	32.2	46.6	59.0	61.4

In Figure 8 we show the results, normalized against the average.

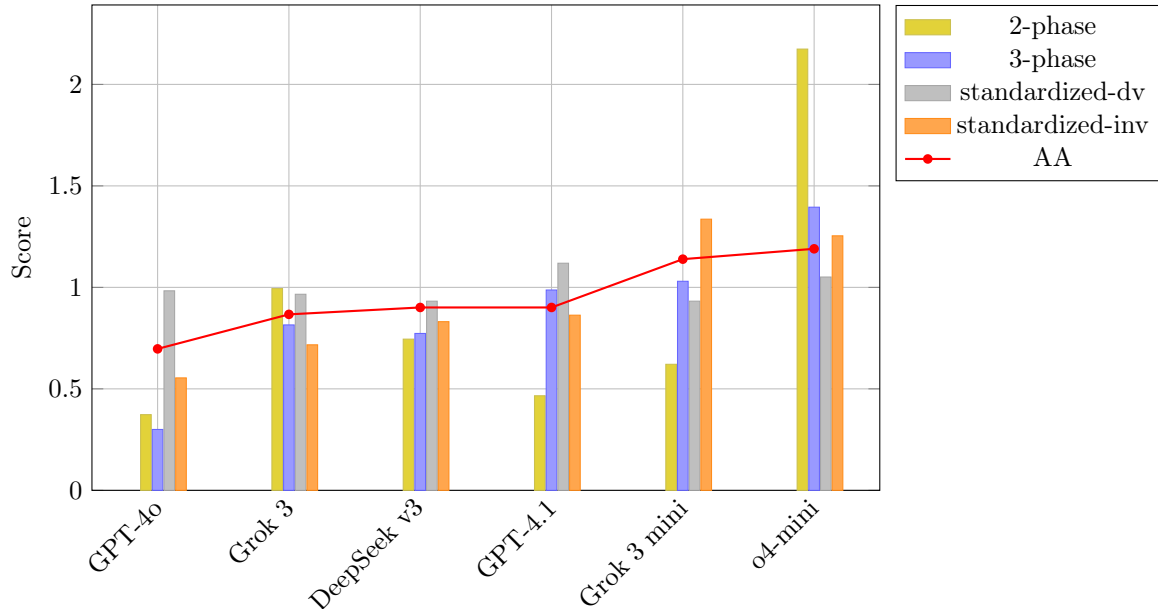


Figure 8: Artificial Analysis scores compared against different SherlockBench evals. Investigation performance is isolated with standardized-dv and d+v-phase performance is isolated with standardized-inv. All scores are normalized against the average for each eval because we want to see trends not absolute numbers.

We can see from this that the AA score is correlated with d+v-phases performance (standardized-inv) but not of inv-phase performance (standardized-dv). This likely indicates that the SherlockBench inv-phase measures a distinct capability.

It also looks like SherlockBench 2-phase is also measuring a distinct capability. Likely related to the models' handling of long context or its propensity to interleave thinking with tool calling.

8 Conclusion

In this paper, we have made three discoveries:

8.1 LLMs investigate worse than random heuristics

When we isolated the LLM's performance on the investigation phase from the decision and verification phases, we discovered that a random investigation is better than the LLM's investigation.

This is surprising and shows that LLM's are not good at generating strategies or behaviours to solve new problems at test-time.

8.2 Two distinct skills

When we compared SherlockBench to the Artificial Analysis Leaderboard, we showed that all LLMs tested had similar and poor performance on the investigation phase, which did not correlate with performance on Artificial Analysis.

Also, performance on the 2-phase test didn't correlate with either the investigation phase performance or with Artificial Analysis.

Therefore we appear to have discovered two distinct skills. This shows that LLMs appear to have several dimensions to their capabilities and cannot be measured by a single scalar value.

8.3 Minor interventions

Twice in this paper we discovered that we could significantly improve the LLM's performance on the test by making minor adjustments to the test conditions. This happened firstly when we compared the 2-phase and 3-phase test, and later when we provided some extra prompting.

This shows that careful software engineering is still required to get good performance out of LLMs.

9 Limitations

9.1 Data Limitation

Due to cost constraints I was not able to test as many models as I would like to, and I was only able to test 10 times per model per problem (I am a solo researcher paying for this work out of pocket).

This results in large error margins, therefore all of the conclusions in this paper should be considered exploratory rather than definitive.

9.2 Investigation Performance

In the section subsection 5.2, we isolate investigation performance by using the same model for decision+verification every time.

This works, but it assumes that the investigation has a scalar quality. In other words we assume there are no idiosyncratic preferences of certain LLMs for certain investigations.

To fix this we would have to re-run this test with every LLM using every other as a verifier.

9.3 Comparison

When writing the section Comparison to Other Benchmarks, it was difficult to find leaderboards which featured all the models we are using. This means we were not able to compare and correlate to a variety of other benchmarks.

9.4 Grok 3 mini

When I attempted to reproduce some of the results for quality control, I found that Grok 3 mini was benchmarking drastically worse than when I first tested it.

xAI does not allow anchoring to specific versions of grok 3 models which means they likely updated this model without changing its version number.

10 Future Research

10.1 Meta-analysis of other benchmarks

In order to determine with more certainty what capabilities SherlockBench is testing, it would be good to perform a broader comparison against more benchmarks.

10.2 Study determinism of different problems

In Figure 3 we saw that the results of solving a given problem with a given model are not very deterministic. It may be interesting to find out if some sorts of problems are more deterministic or random than others.

10.3 Study Success rate by Function by Model

Is a model's success-rate on a single problem predicted by its over-all score on the full problem-set? Or do different models have idiosyncratic patterns of problems they find or easy?

We include a preliminary chart for this in Appendix D, but the small number of samples per model+problem (10) means it isn't reliable.

10.4 Human Baseline

It would be good to test how well humans perform at the benchmark, also isolating performance on each phase.

10.5 Isolate d+v-phases

In this paper we did not isolate d+v-phases (decision&verification) performance from each-other. This is because I consider those parts of the test to be less novel than the inv-phase. However it would be good to do some extra tests to see how often a model that succeeds at the decision goes on to fail at the verifications.

10.6 Prompts

We could investigate how different prompts affect different models performance on SherlockBench.

10.7 More Fine-tuning

We could expand on the fine-tuning experiments to include more models and more training methods.

10.8 Temperature

We could test how temperature affects investigation performance

11 Appendices

11.1 Appendix A: Random Investigation Algorithm

The algorithm for randomly selecting inputs is:

- for integer inputs, select a random int using this Clojure code `(rand-nth [(rand-int 100) (rand-int 20)])`
- for string inputs, select a random dictionary word
- for boolean, randomly select True or False

11.2 Appendix B: Average tool call counts per model on 3-phase test

model	3-phase score	average tool call count
GPT-4o	14	14
DeepSeek v3	36	14
Grok 3	38	9
GPT-4.1	46	21
Grok 3 mini	48	11
o4-mini	65	12

11.3 Appendix C: GPT-4.1 investigation of function "add biggest and smallest"

```
(0, 0, 0) 0
(1, 0, 0) 1
(0, 1, 0) 1
(0, 0, 1) 1
(2, 0, 0) 2
(0, 2, 0) 2
(0, 0, 2) 2
(1, 1, 1) 2
(2, 3, 4) 6
(5, 7, 11) 16
(10, 20, 30) 40
(3, 0, 4) 4
(3, 2, 0) 3
(0, 5, 4) 5
(7, 3, 0) 7
(7, 0, 3) 7
(0, 7, 3) 7
(3, 7, 0) 7
(6, 6, 6) 12
(2, 3, 5) 7
(9, 8, 1)
```

11.4 Appendix D: Chart of performance by Function by Model

In Figure 9 look at how each problem performs across the models as a line chart:

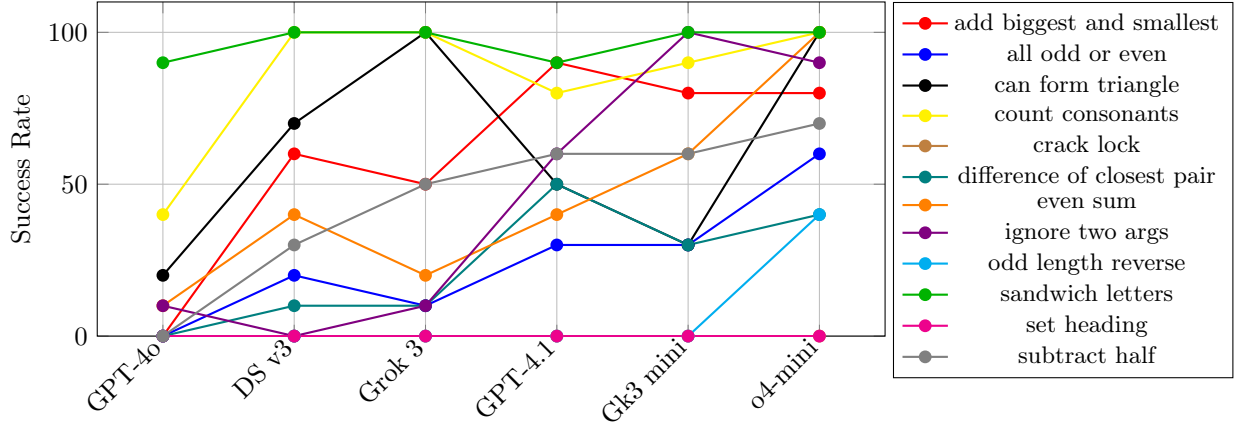


Figure 9: Success Rate per Model, per Function

Due to only having 10 samples per problem per model we cannot take this as accurate. However it does look like certain models are particularly good or bad at certain problems. For example Grok 3 is particularly good at "can form triangle".

11.5 Appendix E: Confidence-Interval Calculation

We report 95% confidence intervals (CIs). Because each bar shows a *proportion of successes*, we use the Wald normal approximation to the binomial distribution:

$$\hat{p} \pm 1.96 \sqrt{\hat{p}(1 - \hat{p})/n}$$

where

- \hat{p} is the observed proportion of successes ($0 \leq \hat{p} \leq 1$);
- n is the number of independent trials that produced that bar (here $n = 120$ for aggregate bars and $n = 10$ for per-problem bars);
- 1.96 is the 97.5th percentile of the standard normal distribution.

The half-width produced by the formula is converted back to percentage points for plotting. When $\hat{p} = 0$ or $\hat{p} = 1$ the Wald formula yields zero width; we keep the interval symmetric around the point estimate in those cases.

11.6 Appendix F: Data Tables

11.6.1 Appendix F.1

Function Name	GPT-4o	DS v3	Grok 3	GPT-4.1	Gk3 mini	o4-mini
add biggest and smallest	0	60	50	90	80	80
all odd or even	0	20	10	30	30	60
can form triangle	20	70	100	50	30	100
count consonants	40	100	100	80	90	100
crack lock	0	0	0	0	0	0
difference of closest pair	0	10	10	50	30	40
even sum	10	40	20	40	60	100
ignore two args	10	0	10	60	100	90
odd length reverse	0	0	0	0	0	40
sandwich letters	90	100	100	90	100	100
set heading	0	0	0	0	0	0
subtract half	0	30	50	60	60	70

11.6.2 Appendix F.2

model	original	standardized-dv
GPT-4o	14	58
DeepSeek v3	36	55
Grok 3	38	57
GPT-4.1	46	66
Grok-3-mini	48	55
o4-mini	65	62
RANDOM	-	77

11.6.3 Appendix F.3

model	original	random-inv	standardized-inv
GPT-4o	14	31	34
DeepSeek v3	36	57	51
Grok 3	38	48	44
GPT-4.1	46	69	53
Grok 3 mini	48	80	82
o4-mini	65	81	77

11.6.4 Appendix F.4

model	original	standardized-dv	random-inv	standardized-inv
GPT-4o	14	58	31	34
DeepSeek v3	36	55	57	51
Grok 3	38	57	48	44
GPT-4.1	46	66	69	53
Grok 3 mini	48	55	80	82
o4-mini	65	62	81	77

11.6.5 Appendix F.5

model	original	assisted
GPT-4o	14	19
DeepSeek v3	36	43
Grok 3	38	39
GPT-4.1	46	45
Grok 3 mini	48	57
o4-mini	65	68

12 Glossary

12.1 Phases

inv-phase

Investigation Phase (phase-1)

d+v-phases

Decision and Verification Phases (phase-2&3)

12.2 Test Modes

2-phase mode

Standard test with investigation+decision combined in phase 1, verification in phase 2

3-phase mode

Test with separate investigation phase, decision phase, and verification phase

inv-vs-dv

Analysis fork that runs investigation phase multiple times, then tests each investigation with decision+verification phases to isolate investigation quality

inv-vs-dv-random

Variant of inv-vs-dv using random investigations as baseline

standardized-dv

Investigation performance isolation fork that uses the same model (o4-mini) for decision+verification phases across all tests

random-inv

Decision+verification isolation fork using random investigations for all problems

standardized-inv

Decision+verification isolation fork using pre-recorded successful investigations

software-assist

Intervention fork that prompts models to continue testing when they stop early with remaining attempts

12.3 Models

DS v3

Deepseek v3

Gk3 mini

Grok 3 mini

12.4 Misc

AA

Artificial Analysis benchmark

LRM

Large Reasoning Model. LLMs which use a chain of thought to work through a problem.

pass@10

what percentage of problems can the model solve at-least once when given 10 attempts per problem

13 Bibliography

Software

- [1] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-api: SherlockBench API*. Version v1.0.1. July 2025. DOI: 10.5281/zenodo.15800941. URL: <https://doi.org/10.5281/zenodo.15800941>.
- [2] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-client: SherlockBench Client*. Version v1.1.0. July 2025. DOI: 10.5281/zenodo.15850651. URL: <https://doi.org/10.5281/zenodo.15850651>.
- [3] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-problems-sherlock1: SherlockBench Sherlock1 Problem-Set*. Version v1.3.1. July 2025. DOI: 10.5281/zenodo.15801057. URL: <https://doi.org/10.5281/zenodo.15801057>.
- [4] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-client-inv-vs-dv: SherlockBench Client inv-vs-dv*. Version v1.1.0. July 2025. DOI: 10.5281/zenodo.15850685. URL: <https://doi.org/10.5281/zenodo.15850685>.
- [5] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-client-inv-vs-dv-random: SherlockBench Client inv-vs-dv-random*. Version v1.1.0. July 2025. DOI: 10.5281/zenodo.15850667. URL: <https://doi.org/10.5281/zenodo.15850667>.
- [6] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-client-standardized-dv: SherlockBench Client standardized-dv*. Version v1.1.0. July 2025. DOI: 10.5281/zenodo.15850663. URL: <https://doi.org/10.5281/zenodo.15850663>.
- [7] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-client-standardized-dv-random: SherlockBench Client standardized-dv-random*. Version v1.1.0. July 2025. DOI: 10.5281/zenodo.15850671. URL: <https://doi.org/10.5281/zenodo.15850671>.
- [8] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-client-random-inv: SherlockBench Client random-inv*. Version v1.1.0. July 2025. DOI: 10.5281/zenodo.15850661. URL: <https://doi.org/10.5281/zenodo.15850661>.
- [9] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-client-standardized-inv: SherlockBench Client standardized-inv*. Version v1.1.0. July 2025. DOI: 10.5281/zenodo.15850697. URL: <https://doi.org/10.5281/zenodo.15850697>.

- [10] Joseph Bruce Anthony Graham. *joseph-xylon/sherlockbench-client-software-assist: SherlockBench Client software-assist*. Version v1.1.0. July 2025. DOI: 10.5281/zenodo.15850666. URL: <https://doi.org/10.5281/zenodo.15850666>.

Data

- [11] Joseph Bruce Anthony Graham. *Example test log for SherlockBench in 2-phase mode*. Zenodo, July 2025. DOI: 10.5281/zenodo.15798073. URL: <https://doi.org/10.5281/zenodo.15798073>.
- [12] Joseph Bruce Anthony Graham. *Example test log for SherlockBench in 3-phase mode*. Zenodo, July 2025. DOI: 10.5281/zenodo.15798128. URL: <https://doi.org/10.5281/zenodo.15798128>.
- [13] Joseph Bruce Anthony Graham. *Json log of SherlockBench Investigation*. Zenodo, July 2025. DOI: 10.5281/zenodo.15798172. URL: <https://doi.org/10.5281/zenodo.15798172>.
- [14] Joseph Bruce Anthony Graham. *SherlockBench test logs - 2-phase mode*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812503. URL: <https://doi.org/10.5281/zenodo.15812503>.
- [15] Joseph Bruce Anthony Graham. *SherlockBench test logs - 3-phase mode*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812505. URL: <https://doi.org/10.5281/zenodo.15812505>.
- [16] Joseph Bruce Anthony Graham. *SherlockBench test logs - inv-vs-dv-random*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812494. URL: <https://doi.org/10.5281/zenodo.15812494>.
- [17] Joseph Bruce Anthony Graham. *SherlockBench test logs - inv-vs-dv*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812482. URL: <https://doi.org/10.5281/zenodo.15812482>.
- [18] Joseph Bruce Anthony Graham. *SherlockBench test logs - standardized-dv-random*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812561. URL: <https://doi.org/10.5281/zenodo.15812561>.
- [19] Joseph Bruce Anthony Graham. *SherlockBench test logs - standardized-dv*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812543. URL: <https://doi.org/10.5281/zenodo.15812543>.
- [20] Joseph Bruce Anthony Graham. *Standardized Investigations*. Zenodo, July 2025. DOI: 10.5281/zenodo.15797843. URL: <https://doi.org/10.5281/zenodo.15797843>.
- [21] Joseph Bruce Anthony Graham. *SherlockBench test logs - standardized-inv*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812569. URL: <https://doi.org/10.5281/zenodo.15812569>.
- [22] Joseph Bruce Anthony Graham. *SherlockBench test logs - random-inv*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812497. URL: <https://doi.org/10.5281/zenodo.15812497>.
- [23] Joseph Bruce Anthony Graham. *SherlockBench test logs - software-assist*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812507. URL: <https://doi.org/10.5281/zenodo.15812507>.
- [24] Joseph Bruce Anthony Graham. *Fine tuning data-set for OpenAI*. Zenodo, July 2025. DOI: 10.5281/zenodo.15797599. URL: <https://doi.org/10.5281/zenodo.15797599>.
- [25] Joseph Bruce Anthony Graham. *SherlockBench test logs - software assist (fine-tuned models)*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812519. URL: <https://doi.org/10.5281/zenodo.15812519>.
- [26] Joseph Bruce Anthony Graham. *SherlockBench test logs - standardized-dv (fine-tuned models)*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812549. URL: <https://doi.org/10.5281/zenodo.15812549>.
- [27] Joseph Bruce Anthony Graham. *SherlockBench test logs - standardized-inv (fine-tuned models)*. Zenodo, July 2025. DOI: 10.5281/zenodo.15812584. URL: <https://doi.org/10.5281/zenodo.15812584>.

Articles

- [28] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8). URL: https://www.cs.cmu.edu/~epxing/Class/10715/reading/Kornick_et_al.pdf.
- [29] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791. URL: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf.
- [30] Yubo Wang et al. *MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark*. 2024. arXiv: 2406.01574 [cs.CL]. URL: <https://arxiv.org/abs/2406.01574>.
- [31] David Rein et al. *GPQA: A Graduate-Level Google-Proof Q&A Benchmark*. 2023. arXiv: 2311.12022 [cs.AI]. URL: <https://arxiv.org/abs/2311.12022>.

- [32] Long Phan et al. *Humanity’s Last Exam*. 2025. arXiv: 2501.14249 [cs.LG]. URL: <https://arxiv.org/abs/2501.14249>.
- [33] Xiao Liu et al. *AgentBench: Evaluating LLMs as Agents*. 2023. arXiv: 2308.03688 [cs.AI]. URL: <https://arxiv.org/abs/2308.03688>.
- [34] Zhicheng Guo et al. *StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models*. 2025. arXiv: 2403.07714 [cs.CL]. URL: <https://arxiv.org/abs/2403.07714>.
- [35] Yun Luo et al. *An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-tuning*. 2025. arXiv: 2308.08747 [cs.CL]. URL: <https://arxiv.org/abs/2308.08747>.
- [36] Artificial Analysis. *LLM Leaderboard: Comparison of Over 100 AI Models*. Continuously updated leaderboard. 2025. URL: <https://artificialanalysis.ai/leaderboards/models> (visited on 07/04/2025).