MEASURING AND REDUCING OVER-RELIANCE ON CONVENIENT EVIDENCE IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents often over-rely on easily inferable information, committing to convenient sources or hints without verifying them. This gullibility is often unrepresented in benchmarks where logic steps are well defined, but abundant in everyday life. We attempt to measure this phenomenon by introducing GulliBench: a set of 50 simple data tasks, each with a planted wrong shortcut and a correct answer that must be derived from the underlying records. The tasks are designed to be well within the coding and reasoning capabilities of frontier models, save for the over-reliance traps. Of the 23 models we tested, even the best one fails on 39% of its attempts, and the other 22 fail on more than half of theirs. We conjecture that this gullibility reflects a behaviour that general capability does not reliably produce, and that it can be learned. By post-training a 35B-parameter LLM with RL on a different set of tasks that expose these contradictions, we improve its accuracy on the benchmark from 3% to 31%. On third-party data-analysis and terminal benchmarks the post-trained model checks its work more often, but this is barely reflected in their scores, and we show through case studies why such benchmarks rarely reward it.

1 INTRODUCTION

LLM agents are already widespread in a variety of professional work, and it is now common to have agents write code, query databases, manipulate data, plot graphs, and write final reports (Handa et al., 2025; Stack Overflow, 2025). Yet these agents frequently fail at completing long end-to-end workflows fully autonomously, and human supervision is still required to catch their errors (Wang et al., 2026; Rabanser et al., 2026; Khanal et al., 2026). At the same time, recent benchmarks show that frontier models have most of the skills and knowledge this work calls for, solving most of the hard terminal tasks of Terminal-Bench 2.1 (Merrill et al., 2026), and scoring comparably on text-to-SQL, mathematics and domain-knowledge benchmarks (Li et al., 2023; Lei et al., 2025; Hendrycks et al., 2021; Glazer et al., 2024; Rein et al., 2024; Phan et al., 2025).

The tasks in those benchmarks, however, usually give the agent information it can rely on, whereas in real work information is often messy and some of what an agent is given is stale, partial, or wrong. Doing the job well then includes weighing the given information against other evidence before committing to an answer. We study what happens when agents skip this step, a failure mode we call *over-reliance*. An over-reliant agent commits to whatever makes an answer readily available, be it a stored value, a statement in the task description, or one of its own earlier results, without verifying it against other evidence in front of it. Figure 1 shows an example, in which stored account balances have gone stale while the ledger they were computed from has not.

Existing benchmarks (e.g., Li et al., 2023; Egg et al., 2025; Merrill et al., 2026) are built to test other skills, so a convenient wrong route appears in them only by accident. To help close this gap we introduce GulliBench, a benchmark that measures whether an agent takes the correct route when the data also offers a convenient but wrong one. The tasks are deliberately easy, so that the measurement is not confounded with the ability of the model (§2). We evaluate 23 frontier models on GulliBench (Figure 2, §3). The best passes 61% of its attempts and most pass fewer than a quarter of theirs. These pass rates are not well explained by capability as it is usually understood, since they do not improve consistently with newer versions or with larger models within a family, and did not improve with more reasoning in the one model we tested at two settings.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Question. What is the total balance held across all accounts? Answer with a single number rounded to 2 decimals.

accounts

account_id	balance
A-01	120.00
A-02	80.00
A-03	45.00

transactions

txn_id	account_id	amount
T-001	A-01	100.00
T-002	A-01	20.00
T-003	A-02	80.00
T-004	A-03	45.00
T-005	A-01	−30.00

Convenient route.
`SUM(accounts.balance)`
= 245.00

Correct route.
`SUM(transactions.amount)`
= 215.00

Figure 1: An example task. The stored balance of account A-01 is stale: the ledger of transactions, from which the balances were once computed, no longer adds up to it. Summing the stored column answers the question in one query and is wrong; summing the ledger is also one query and is right. The question carries no hint that the stored column should be checked.

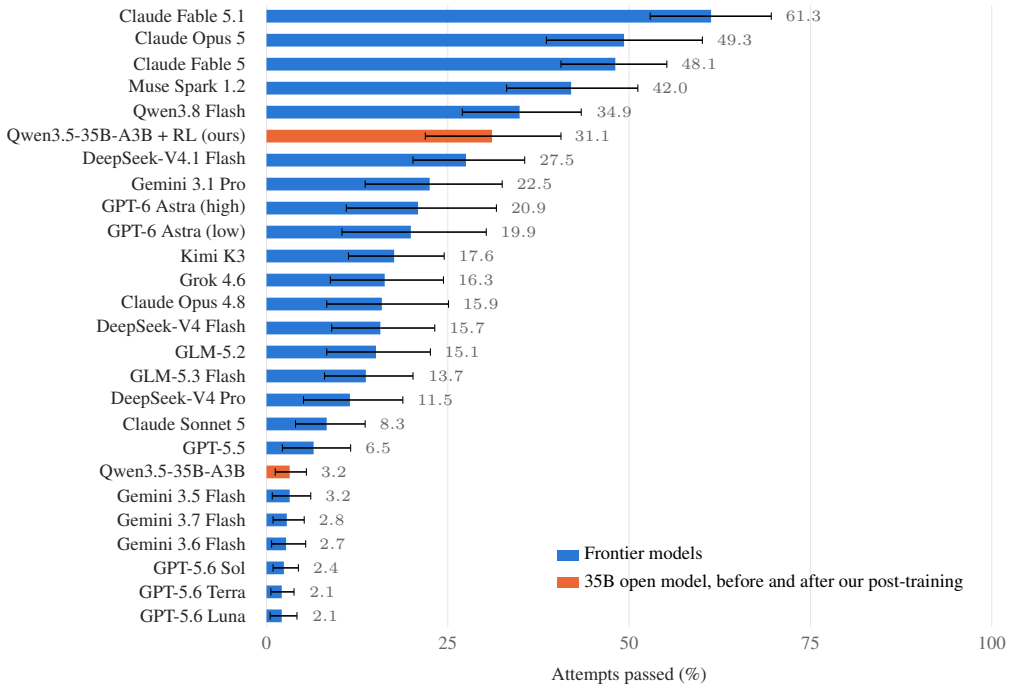


Figure 2: Share of attempts passed on GulliBench (150 tasks, 5 attempts each) by the 23 frontier models (GPT-6 Astra at two reasoning efforts) and by an open 35B model before and after our post-training, all in the same harness (Table E.1). Error bars are 95% bootstrap intervals over the 50 pairs.

We therefore conjecture that checking is a behaviour that general capability does not reliably produce, and that it can be learned. To test this we post-train an open 35B-parameter model with reinforcement learning on a separate set of tasks built around different shortcut types from those in the evaluation set, so that its gains cannot come from memorising the benchmark (§4). On GulliBench its pass rate rises from 3.2% to 31.1%, above 18 of the 23 frontier models (Figure 2).

Finally, we ask if what the model learned generalizes beyond GulliBench (§5). We compare the base and post-trained models on four third-party benchmarks of data analysis and coding (Li et al., 2023; Egg et al., 2025; Merrill et al., 2026; Deng et al., 2025). On three of them the post-trained model checks its work more often, but its scores barely change. Through case studies we find that on most tasks the extra checks would not change the answer, and on some they lead to a plausible reading of the task that the benchmark marks as wrong. We take this as evidence that on most tasks of these benchmarks, built to measure other skills, checking makes little difference to the score.

Contributions.

1. We introduce GulliBench, a benchmark of simple data tasks in which the data offers a convenient but wrong route to the answer, with a generator, a fixed evaluation set in three data formats, and grading by comparison with a known answer. We release all tasks, data, and code publicly (anonymised for review at `osf.io/p9dzj`).
2. We evaluate 23 frontier models and show that most take the wrong route on most tasks, and that this is not explained by model version or model size, nor, in the one model we tested at two settings, by reasoning effort.
3. We show that the behaviour can be learned, by post-training a 35B model on tasks with different shortcuts and raising its pass rate from 3.2% to 31.1%, above most of the frontier models.
4. We show that the learned behaviour carries to three of four third-party benchmarks, and present case studies of why their scores barely register it.

2 Measuring over-reliance with planted shortcuts

We measure over-reliance, as defined in §1, with data tasks in which the data itself offers several routes to an answer. This section describes how such tasks are generated, what makes them fair, and how GulliBench, the fixed evaluation set used in the rest of the paper, is composed from them.

2.1 Task anatomy

A task consists of a small dataset of a few dozen records, usually split between two tables, and a one-sentence question with a single numeric answer. The generator knows the correct answer and the answer that each convenient route gives, and it discards any instance in which those coincide, so by construction each convenient route gives a different answer from the correct one. The task question is purposefully left bare, with no hints or instructions telling the agent to distrust the data or to check it. The environment contains nothing but the dataset; the agent is given a set of tools to inspect it and must submit a single number as its answer. Grading is by exact comparison of the submitted number with the correct answer, up to a small tolerance. Figure 1 shows an example task, and Appendix A.1 a complete one from the evaluation set.

2.2 The catalogue of shortcut types

Our generator has a catalogue of 61 *shortcut types*, each pairing a query operation with a way in which data commonly misleads it. Most involve a stored value that is stale, wrong, or recorded at the wrong grain. Others are pitfalls of the data itself, such as numbers stored as text, placeholder values or mixed units. Appendix A lists the families with an example of each, and the full catalogue.

The catalogue was assembled by systematically mapping common query operations, such as filtering and joining, and asking frontier models from several providers to brainstorm ways in which they could fail. Each resulting candidate then had to pass two rules to be added to the catalogue. First, a shortcut type is admitted only if the data can manufacture *one* tempting wrong answer, deterministically. Errors that have no such answer are excluded. Second, the correct route must be discoverable from the data alone, meaning that the convenient value is contradicted by records the agent can read, so that a check comparing the two would reveal the problem. A shortcut type whose only tell is domain knowledge, or whose question has more than one reasonable reading, is not admitted.

Each shortcut type can be rendered in different domains and with different data while the computation stays the same, which gives many distinct instances of each type (§4).

2.3 Harness and formats

The agent reaches the data only through the tools of a purpose-built MCP (Model Context Protocol) server (Anthropic, 2024). We wrote our own tool-calling agent rather than using a popular terminal-based one, so that we control how the model reaches the data, and so that the results do not partly reflect training on familiar scaffolds. The agent and the server are released with the benchmark.

Within this harness, each task instance is rendered in three formats. Models are trained extensively on writing SQL and on navigating file systems, and rendering the same data in different formats lets us test whether that training plays a role in the behaviour (§3). In the *table* format the agent can only inspect the tables, listing columns and paging through rows, so it must do any filtering or arithmetic itself; in the *documents* format each record is a text file it can list, read and search; and in the *SQL* format it queries a database. The correct answer and the convenient routes are the same in all three.

2.4 THE GULLIBENCH EVALUATION SET

The GulliBench evaluation set consists of 50 tasks rendered in the three formats, 150 task instances in all. Unlike the example of Figure 1, which asks one question over one table and plants one shortcut, each of these tasks has two steps. The agent first has to select an entity from one set of records and then compute a value for it from another, and each step has a shortcut of its own. We call these *two-step* tasks. We use two steps because that is how most real questions get answered, by finding something and then computing over it, and because the result of the first step becomes an input to the second that the agent can either trust or check, which a single-step task cannot test. Each step is still one simple query, and the second asks for nothing the first did not.

To build the evaluation set we adapted 19 catalogue types into ten selection and ten aggregation shortcuts and composed them pairwise into 100 two-step tasks, in the three formats. Seven pairs were dropped as ambiguous, leaving 93. All 93 could serve as the benchmark, but the strongest model already passed more than half of them nearly every time, and a set diluted with tasks that the best models no longer fail discriminates poorly at the top, so we selected the harder half. We first ran a sample of 20 of these on a panel of fourteen frontier models from different families, as available on 9 August 2026 (Appendix B), to find the strongest, which was Claude Fable 5 by a wide margin. We then used that model to select the benchmark adversarially: we ran it on all 93 pairs, five attempts per task and format, and kept the 50 pairs it passed least often.

Selecting tasks on one model's failures biases the benchmark against that model and, to the extent that models fail the same tasks, against every other model as well. Of the 20 pairs run on the panel, 12 were kept and 8 excluded, and every model above the floor scores lower on the kept pairs, so the selection harmed all of the models and not only the one used to select. The selection was fixed on 9 August 2026, and ten of our 24 entries came later, including three of the five best, so they were not selected against directly.

A low pass rate is evidence of over-reliance only if every task is fair, with no second reasonable reading under which the convenient value is right. In our tasks the convenient value is either derived from the records, as a stored balance, rank or flag is, or a misreading of them, such as a number stored as text, so when the two disagree the records are authoritative unless the task itself is ambiguous. The data also shows the agent which source to trust, since in every evaluation task the stored aggregate agrees with the records for every entity except the two the shortcuts involve (200 of 200 across the 50 pairs). Like any data task, ours take the records the agent can read as the data; under a reading in which those records are incomplete in ways the data does not show, the task would have no answer. We check for ambiguity in three ways. First, the admission rules of §2.2 are applied when a shortcut type is designed, so the convenient value is always contradicted by records the agent can read. Second, every generated instance must pass two scripted checks in every format, in which a policy that derives the answer from the records must be graded correct and one that trusts the convenient value must be graded wrong. Finally, before a two-step pair enters the benchmark, a model given a prompt tuned to elicit careful analysis must reach the correct answer reliably; we manually reviewed every pair on which it did not, and dropped those that were ambiguous (Appendix C gives the prompt). Together, these checks provide evidence that the correct answer can be reached unambiguously. They rejected several candidate tasks during construction (Appendix D gives two examples), but we cannot guarantee that every remaining task is fair. For this reason we release the benchmark, the generator and the checks, so that every task can be independently inspected.

3 FRONTIER AGENTS OVER-RELY

We evaluate 23 frontier models on GulliBench, in 24 configurations, five attempts per task, through the harness of §2.3 with a limit of 30 turns, and with each provider's default settings except where a

reasoning-effort level is named. Intervals throughout are 95% bootstrap intervals that resample the 50 pairs, keeping each pair's three formats and five attempts together, since these are not independent; Appendix E.6 gives intervals that resample shortcut types instead.

Figure 2 shows the share of attempts on which each model reaches the correct answer; the numbers behind it, with the breakdown of each model's failures, are in Appendix E. The confidence intervals of neighbouring models overlap, so the figure gives tiers rather than a strict ranking, with one model above 60%, a small group near half, a broad middle between 10% and 35%, and a floor near zero.

Most failures are shortcuts. Of the attempts that fail across the panel, 93.6% give exactly a shortcut answer. The remaining 6.4% are split between other wrong answers (4.9%), mostly a single arithmetic pitfall on one group of tasks, and attempts with no answer at all (1.5%). Of the attempts that take a shortcut, 98% take the second-step shortcut, alone or together with the first. In 79% of the attempts that report a stored aggregate, the correct value never appears in any tool output, so these failures are not a disagreement about which source is right (Appendix E.7). Analysing the failures by shortcut, models almost never err at the selection step, where finding the entity means reading the very records that hold the anomaly, whereas at the aggregation step their error rate ranges from 26% to 92%. It is lowest when the question names an operation the model should perform, such as a count, and highest when it names a state of the entity, such as its current balance (Table E.2).

The tasks are within the models' ability. In every format failed attempts are no longer than successful ones, and what they lack is the check that would have exposed the shortcut (Table E.3). That check is within reach of even low-cost models, one of which reaches the correct answer on 87% of its runs when given the data directly and told to reconcile stored values with the underlying records (Appendix C).

Presentation does not matter, on average. Pooled over the panel, pass rates in the three formats differ by less than about three points. This is evidence that proficiency with the tool does not change the behaviour. Models are trained extensively on SQL, which answers each task in one statement, yet they do no better there than in the table format, where they have to filter and add up the records themselves. Individual models do differ by format, in both directions (Appendix E.4).

Pass rates only loosely follow capability. On the 12 entries a public capability index covers (Artificial Analysis, 2026), its rank correlation with our pass rates is weak ($\rho = 0.26$, 95% CI $[-0.40, 0.85]$) and vanishes without Claude Fable 5.1. Within a family the larger model is not reliably above the smaller one, and across families models regarded as stronger rank below models regarded as weaker; the clearest case is Qwen3.8 Flash, a small low-cost model, which ranks above every GPT and Gemini entry. Newer generations are not reliably better either, as GPT-5.5 scores above every GPT-5.6 tier and Sonnet 5 below the older Opus 4.8. More reasoning does not seem to help either, as GPT-6 Astra, the one model we ran at two settings, scores about the same at low and high reasoning effort (19.9% and 20.9%).

At the same time, the latest model from each of two providers passes far more often than its predecessor. Claude Fable 5.1 scores well above Fable 5, and GPT-6 Astra far above every GPT-5.6 tier. Even so, the two remain forty points apart on GulliBench, although they are close on general capability benchmarks (Marten, 2026; Lu et al., 2026), and the much smaller Qwen3.8 Flash sits between them. We take this as evidence that the skill can be improved, as it has been in some recent models, but that it does not reliably come with general capability and has to be targeted directly (§4).

4 Reducing over-reliance by post-training

The results of §3 suggest that general capability does not reliably remove over-reliance. In this section we test whether it can be reduced by training. We post-train a small open model with reinforcement learning on tasks generated by the same method, and ask whether the gain carries to GulliBench, whose shortcut types the model never sees in training.

4.1 Training tasks

The 19 catalogue types behind the GulliBench shortcuts (§2.4) are held out of training, together with every two-step task built from any of them, and training uses only the other 42 types (Appendix A marks which is which). From these 42 shortcut types we generate 3,780 single-step tasks, one for

each type in ten domains with three random seeds each, rendered in the three formats ($42\times10\times3\times3$), and 2,700 two-step tasks from 30 pairs composed from training-side selection and aggregation types only (Table A.3), rendered in the same domains and with the same seeds ($30 \times 10 \times 3 \times 3$), for 6,480 training tasks in all, which we release with the benchmark. Since the evaluation set shares neither its shortcut types nor its data with the training set, and the training domains are varied, a gain on GulliBench cannot come from memorising its shortcuts or data, although some training types, like the stale balance of Figure 1, share the stale-aggregate pattern of most evaluation shortcuts.

4.2 Method

We post-train Qwen3.5-35B-A3B (Qwen Team, 2026a), an open mixture-of-experts model with 35B total and 3B active parameters, with GRPO (Shao et al., 2024). Rollouts run in a terminal agent scaffold, Terminus 2 (Merrill et al., 2026), with a limit of 16 turns of at most 8,192 tokens each, and the reward is 1 if the submitted answer is correct and 0 otherwise, from the same grader as the benchmark. Each step draws 16 tasks from the pool of §4.1 without repetition and samples 8 rollouts per task; training is asynchronous, with rollouts from a policy at most 4 steps stale, corrected by importance sampling. We stopped after 192 steps, having used 3,072 of the 6,480 tasks, when the compute budget ran out, although the model was still improving (Appendix F, which also gives every setting of the run); no checkpoint or hyperparameter was chosen using GulliBench.

4.3 Results

The pass rate on GulliBench rises with training. In the harness it was trained on, the model goes from passing 2.7% of attempts to 58.9% after 192 steps (Figure F.1). The gain is reflected in a harness it never saw, our agent of §2.3, where its pass rate goes from 3.2% to 31.1% (95% CI $[21.9\%, 40.5\%]$). That would place it sixth among the 23 frontier models of Figure 2, although it is far smaller than most of them. The harness gives it no advantage, since it was trained in a different scaffold and never saw ours, although unlike the frontier models it was trained on tasks from our generator. The gain is smaller than under the training scaffold, since part of what the model learned is how to use that scaffold.

The model takes fewer shortcuts. Classifying the post-trained model's attempts as in §3 (Table E.1), the share that land on a shortcut answer falls from 89% to 58%, while other wrong answers rise slightly, from 7.5% to 10.9%. The gain is at the aggregation step, where the model reports fewer stored values, and does not reach held-out pitfalls of recomputation or selection (Table E.5).

The gain is uneven across formats, largest on documents (45.2%) and smallest on SQL (22.8%), where the base model is flat at 3.2% in all three (Table E.4). A likely reason is that the documents format forces the model to read the records one by one, so a model disposed to check meets the records that contradict the stored value in the course of its work.

Prompting does not substitute for training. Telling the base model to be skeptical and to check stored values against the underlying records, with a prompt similar to the one that lets the low-cost model of §3 solve almost every task, has no measurable effect on its pass rate, 3.9% against 3.2% (paired difference $+0.7$ points, 95% CI $[-0.9, +2.5]$). This suggests that an instruction to check is not enough for this model, whereas training is.

Taken together, these results suggest that the model learned a behaviour that carries beyond the tasks it was trained on, though only where it reads stored values. However, all of this is measured on tasks generated by our own method, and it does not show that the behaviour carries to tasks of other kinds. In the next section we test this on four third-party benchmarks.

5 Does the behaviour carry to other tasks?

To test whether the post-trained model checks its work beyond tasks of our own making, we compare it with the base model on four third-party benchmarks, none of which we used in post-training. Two are data benchmarks, BIRD (Li et al., 2023), a text-to-SQL benchmark in which the agent answers a question over a database and is given a hint, and DABstep (Egg et al., 2025), a data-analysis benchmark in which the agent answers a question from data files and a manual. We expect data tasks to offer the most occasions for the behaviour to show, since real data and documentation are

		Base	Post-trained	Δ	95% CI
GulliBench (150 tasks)	Score	3.2	31.1	+27.9	[+18.9, +37.1]
	Re-checks (level ≥ 1)	86.5	98.5	+12.0	[+7.9, +16.5]
	Tests a premise (level 2)	45.1	80.8	+35.7	[+26.0, +44.8]
DABstep (450 tasks)	Score	25.8	30.2	+4.4	[+2.5, +6.2]
	Re-checks (level ≥ 1)	46.9	92.4	+45.4	[+42.5, +48.3]
	Tests a premise (level 2)	26.1	39.6	+13.5	[+10.6, +16.4]
BIRD (150 tasks)	Score	68.8	66.7	−2.1	[−4.8, +0.4]
	Re-checks (level ≥ 1)	14.8	40.1	+25.3	[+20.1, +30.7]
	Tests a premise (level 2)	4.8	15.1	+10.3	[+6.4, +14.4]
Terminal-Bench 2.1 (30 tasks)	Score	48.9	52.2	+3.3	[−5.6, +12.2]
	Re-checks (level ≥ 1)	58.9	74.4	+15.6	[+3.3, +28.9]
	Tests a premise (level 2)	8.9	22.2	+13.3	[+1.1, +26.7]
SWE-bench Pro (100 tasks)	Score	37.3	39.0	+1.7	[−4.0, +7.2]
	Re-checks (level ≥ 1)	54.0	60.2	+6.2	[+0.0, +12.2]
	Tests a premise (level 2)	14.3	17.3	+3.0	[−1.7, +7.7]

Table 1: Base and post-trained model on each evaluation set: share of attempts rewarded, share graded at level 1 or above, and share graded at level 2, in percent, with the task-paired difference and its 95% bootstrap interval over tasks (for GulliBench, over its 50 pairs, each with its three formats). GulliBench, DABstep and BIRD have five attempts per task, the two coding benchmarks three. GulliBench is run under our own agent, as in Figure 2, and the others under Terminus 2.

full of values that can be taken as given. To test whether the behaviour carries over to tasks other than data analysis, we also use a sample from two terminal and coding benchmarks, Terminal-Bench 2.1 (Merrill et al., 2026) and SWE-bench Pro (Deng et al., 2025). We use 150 BIRD tasks and all 450 DABstep tasks with 5 attempts each, and 30 Terminal-Bench and 100 SWE-bench Pro tasks with three attempts each, all under the same terminal agent, Terminus 2 (Merrill et al., 2026).

5.1 Reading behaviour off the trajectories

While a benchmark's scores tell us whether the models can correctly reach the expected answer, to properly understand the changes in behaviour we need to read the trajectories and decide whether the agent tested something its answer relied on, rather than taking it as given. This is hard to determine without clear guidelines, since the same action can be a check or not depending on what preceded it.

Therefore, to grade reliably and consistently, we created a grading scheme based on three levels. At level 0 the agent commits to its answer without testing anything. At level 1 it re-checks along the same route: it redoes or re-derives a computation, so it is questioning its current approach without proposing another one. At level 2 it tests a premise by a different route, one that could have sent it to a different approach or answer. We also excluded a few common ways in which a model appears to re-check but that do not reflect the behaviour we want to measure, namely reacting to an error, re-running the same command unchanged, and checking the format of the output.

We apply this definition with LLM judges (Zheng et al., 2023) to the 8,272 trajectories we obtained from running the two models on the benchmarks. To calibrate these judges, one of the authors hand-graded 52 trajectories sampled across the benchmarks, blind to which model produced each one and to its outcome. We use these labelled trajectories to calibrate a panel of judges (Verga et al., 2024). Two judges from different model families grade every trajectory, and a third grades the ones in which they disagree, the end result being the median of the three. The judges are given only the rules, the task and the trajectory, with no access to the model under test or to the task's reward. On these trajectories, which we also used to refine the definition, the panel agrees with the human on every gradable one. The rules, the calibration sampling and the detailed results are in Appendix G.

5.2 Results

The learned behaviour carries to three of the four benchmarks. Table 1 shows that on BIRD, DABstep and Terminal-Bench the post-trained model re-checks its work and tests premises far more

often than the base model, whereas on SWE-bench Pro re-checks rise only slightly and premise tests by an amount within noise. We cannot tell whether the behaviour carries less to this kind of work or our grading does not capture it there. On the two coding benchmarks the judges disagree far more often, on 36% and 49% of trajectories against 19% elsewhere, and our calibration set contains no repository-editing trajectories (Appendix G); most checking on such tasks is running the tests, which the definition places at level 1.

The extra checking rarely changes the scores. Table 1 also shows that the scores barely move, with a clear but modest gain on DABstep and changes within noise elsewhere, although they do not fall measurably either, so post-training had no measurable cost to skills the model already had. On BIRD and DABstep, level-2 attempts rise as much on the tasks whose score fell or stayed level as on those whose score rose ($r = -0.05$ and $+0.01$ across tasks), whereas on GulliBench they rise most where the score rose (49 against 28 points). Within a task, level-2 attempts score no better than the others on BIRD and the two coding benchmarks and only slightly better on DABstep ($+3.6$ points), against $+9.5$ on GulliBench, and level-1 attempts score no better than level-0 anywhere.

5.3 Why the scores barely move

To understand why the scores barely move, we studied tasks on which the two models behave differently. We present three cases, each showing a different way in which a check can go unrewarded. Where a case depends on whether an answer is correct, we verified the answer by running the queries on the benchmark's own data. The rules by which we selected the cases are in Appendix H.

A check can leave the answer unchanged. A BIRD task asks for a count of posts with no comments, and its hint points to a stored comment count. Every attempt of both models answers from the stored count and is rewarded. In four of its five attempts, however, the post-trained model first checks the stored count against the comments themselves before submitting. The count turns out to be correct, so the answer and the score stay the same. Across BIRD, the post-trained model reaches level 2 more often on 34 of the 121 tasks on which the two models score the same, and on DABstep on 150 of 307.

A check can be penalised when the reference answer is wrong. A BIRD task asks which home team lost the fewest matches in a season, but its reference answer does not count teams that never lost at home, of which there are three. In four of five attempts the base model follows the reference and is marked correct, while the post-trained model checks for teams that never lost at home and is marked wrong. This is one of six BIRD tasks on which the post-trained model's level-2 attempts score lower than its others, and each of the six has a problem with the task itself, a wrong reference answer in three and an ambiguous question or hint in the other three (Appendix H), in line with independent audits of BIRD (Wretblad et al., 2024; Jin et al., 2026).

A binary score does not register partial progress. A DABstep task asks which fee rules apply to a merchant on a given day, which requires checking each of the day's transactions against the conditions in the manual. The shortcut is to apply only the conditions that concern the merchant, which returns many more rules than the correct answer. The base model takes this shortcut in four of its five attempts. The post-trained model goes past it in three of its five attempts, but then filters wrongly and drops some of the correct rules. All ten attempts score zero, so the score does not show that the post-trained model leaves the shortcut three times as often.

The behaviour learned on GulliBench carries to most of these benchmarks, but their scores register it only on the few tasks that turn on it; on most it makes no difference, and on a few it is penalised.

6 Discussion

Our results suggest that current models largely lack the habit of checking convenient evidence before relying on it, and that this habit does not reliably come with general capability. All but one of the frontier models fail more than half of their attempts at our easy tasks, almost always by taking a convenient route, and widely used data and coding benchmarks barely register the behaviour, so it cannot be relied on to improve as a side effect of other training. Yet a small open model trained on generated tasks learns it, and carries it to third-party data. Since agents now work on data with exactly the problems we plant, we read this as a case for measuring and training it directly.

The evidence for this case, however, comes from few settings. The benchmark uses small generated datasets in which other records always contradict the misleading value. An agent that never trusts a stored value would therefore pass every task, which is the behaviour we set out to measure, since deriving a value from the records costs a query and is right whether or not the stored value is, and a high score shows that an agent avoids this error, not that its trust in evidence is calibrated in general. For the same reason we do not add control tasks. Tasks in which the stored value is correct are passed alike by agents that check and agents that do not. A control in which checking leads astray would need records that are incomplete, such as a ledger missing some of the postings behind a balance. If the gap is hidden, the data cannot establish the answer, and if it is visible, as when the ledger starts after the account was opened, the gap is itself a cue to check, which our tasks deliberately withhold, and the task would test the opposite failure, distrusting a value that is right, rather than over-reliance. Real data is larger and need not contradict a stale value, which should make the behaviour both more consequential and harder to measure, so the next step is to extend the benchmark to larger, messier data and to tasks beyond data analysis. Beyond the data, every model runs in our own harness at default settings, the training result rests on one run of one model, and our transfer evaluation covers four benchmarks, one of them on a 30-task subset. On the one that edits a code repository, we cannot yet tell whether the behaviour carries less or our grading misses it. Finally, on the third-party benchmarks we show a change in behaviour rather than in outcome, and showing that checking pays would need tasks with known discrepancies. We release all tasks, data, code and trajectories, and will release the post-trained weights with the final version.

7 Related work

Benchmarks for data and agentic work. Data agents are evaluated on questions over databases and files (Li et al., 2023; Lei et al., 2025; Egg et al., 2025), and broader agentic benchmarks cover terminal and software work (Merrill et al., 2026). These benchmarks score only the final answer and contain misleading values only by accident, so they cannot attribute a failure to over-reliance.

Over-trusting given information. Models defer to what they are told, adopting a user's stated view (Sharma et al., 2024), following hints in the prompt without acknowledging them (Chen et al., 2025), and being led astray by irrelevant context (Mirzadeh et al., 2025). Closest to our setting, agents often adopt corrupted tool outputs (Yang et al., 2026) and struggle to recover from faults injected into their environment (Tian et al., 2026). Over tables, models fall back on prior knowledge when the data contradicts familiar facts (Wang & Liu, 2026), which is the opposite failure to ours, where the agent trusts a value in the data too much. In all of this work the misleading content is asserted by a user, a hint, a tool or the model's prior. In GulliBench nothing false is asserted, the wrong value being an ordinary field of the data that only a computation the agent chooses to make will reveal.

Planted shortcuts. HANS measures shortcut learning with examples on which a known heuristic fails (McCoy et al., 2019), and ImpossibleBench tests whether coding agents exploit tests that contradict their specification (Zhong et al., 2025). We use the same construction, but our planted route is a plausible answer rather than a flaw in the grader.

Training behaviours with reinforcement learning. Outcome rewards can bring out verification and backtracking in reasoning models (DeepSeek-AI, 2025; Gandhi et al., 2025) and multi-turn agents (Jin et al., 2025; Wang et al., 2025). Whether this teaches new behaviour or makes existing behaviour more frequent is debated (Yue et al., 2025); since our base model already tests premises on some attempts (Table 1), our result is consistent with the second.

8 Conclusion

We introduced GulliBench, a benchmark of simple data tasks in which the data offers a convenient but wrong route to the answer. All but one of the 23 frontier models we evaluated fail more than half of their attempts, almost always by taking a planted shortcut, and their pass rates follow general capability only loosely. Post-training a small open model on tasks with disjoint shortcut types raises its pass rate from 3.2% to 31.1%, and the model checks its work more often on three of four third-party benchmarks, whose scores change little, except for a modest gain on DABstep. These results suggest that checking given evidence is a behaviour that capability alone does not reliably produce, and that it needs to be measured and trained directly.

AI USE STATEMENT

We used generative AI tools in several parts of this work. *Synthetic data.* The benchmark and training tasks are produced by deterministic code, but LLMs contributed to the data behind them. Frontier models from several providers were asked to brainstorm ways in which common query operations can fail, and some of the shortcut types in the catalogue originate there (§2.2); LLM agents also wrote the domain renderings of each shortcut type. Every shortcut type and rendering had to pass the scripted checks of §2.4 before use. *Hypotheses, methodology and experiments.* The idea for the benchmark, its design, the hypotheses we test and the experiments came mostly from the authors. We used AI assistants for feedback on the methodology and experimental design, and to help brainstorm and refine ideas, and the authors decided what to adopt. *Implementation.* The generator, the agent harness, the classification of answers and the analysis scripts were written with the help of AI coding assistants. All code was tested by the authors, the classification of answers was checked against the stored rewards, and the expected answers of the tasks we discuss were verified by executing their queries. *Qualitative analysis and interpretation.* The checking levels of §5 are assigned by LLM judges, calibrated against trajectories graded by one of the authors (Appendix G). AI assistants also helped read trajectories for the case studies and helped interpret results, in an iterative process in which the authors directed what to look for and what to analyse, and checked the case studies and interpretations reported in the paper against the underlying data. *Writing and literature.* We used AI assistants to draft and edit parts of the text, to search for related work and to format references. The text was developed through an extensive iterative process in which the authors wrote and rewrote most of the paper themselves, and every cited reference was checked against its published version. We did not use generative AI to formulate mathematical claims or proofs, and translation is not applicable to this work. We have reviewed all AI-assisted work and take responsibility for the final content of this paper, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves no human subjects beyond the authors, who graded agent trajectories themselves. The tasks of GulliBench and our training tasks are synthetic and contain no personal data. We evaluated commercial models through their providers' interfaces under the providers' terms, and used the third-party benchmarks of §5 under their licences; the trajectories we release that quote those benchmarks remain under their licences. Released trajectories are scrubbed of credentials and of identifiers of our infrastructure. We release code under the MIT licence and data under CC BY 4.0. Because we release the generator for every shortcut type, including those of the evaluation set, models could in principle be trained on the evaluation's shortcut types; results on GulliBench should therefore be reported together with a statement of whether the model was trained on data from the generator. The behaviour we study, checking given evidence before relying on it, is one that makes agents more reliable, and we see no direct route from our findings or released artifacts to harm.

REPRODUCIBILITY STATEMENT

Section 2 describes how the tasks are generated, checked and selected, with the full catalogue of shortcut types in Appendix A, the selection panel in Appendix B and the careful-analyst prompt in Appendix C. The evaluation setting (harness, limits and attempts) is given at the start of §3, and Appendix E reports every model's results with confidence intervals, together with the per-shortcut, per-format and tool-call breakdowns. The training data and setup are in §4.1 and §4.2, with the training curve in Appendix F. The grading scheme, judge panel and calibration of §5.1 are detailed in Appendix G, and the case studies, with their selection rules, in Appendix H. The supplementary material contains the evaluation set and the other screened tasks, the generator and its checks, the agent harness, the training tasks with the scripts that regenerate them, every model's per-attempt results, the judge prompt and grades, and the scripts that produce every number, table and figure in the paper; all bootstrap intervals use fixed seeds. The scrubbed trajectories and the rendered training tasks, too large for the supplementary archive, are at `https://osf.io/p9dzj/overview?view_only=17ba0502388844f9ab49dabf9b5f7385`. The weights of the post-trained model will be released with the final version of the paper.

REFERENCES

Anthropic. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol, 2024. Accessed 2026-09-23.

Anthropic. Claude Fable 5 and Claude Mythos 5. https://www.anthropic.com/news/claude-fable-5-mythos-5, 2026a.

Anthropic. Claude Fable 5.1 and Claude Mythos 5.1 system card. https://www.anthropic.com/claude-fable-and-mythos-5-1, 2026b.

Anthropic. Introducing Claude Opus 4.8. https://www.anthropic.com/news/claude-opus-4-8, 2026c.

Anthropic. Claude Opus 5 system card. https://www-cdn.anthropic.com/c5fbac3f0b1280a933ebd26d3cb8bb9f5bdeaf48/Claude%20Opus%205%20System%20Card.pdf, 2026d.

Anthropic. Claude Sonnet 5 system card. https://www.anthropic.com/claude-sonnet-5-system-card, 2026e.

Artificial Analysis. Artificial analysis intelligence index. https://artificialanalysis.ai/leaderboards/models, 2026. Accessed 2026-09-25.

Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, et al. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*, 2025.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *Nature*, 645, 2025. arXiv:2501.12948.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026a.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026b.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. DABstep: Data agent benchmark for multi-step reasoning. *arXiv preprint arXiv:2506.23719*, 2025.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STaRs. *arXiv preprint arXiv:2503.01307*, 2025.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, et al. FrontierMath: A benchmark for evaluating advanced mathematical reasoning in AI. *arXiv preprint arXiv:2411.04872*, 2024.

Google. What's new in Gemini 3.5 Flash. https://ai.google.dev/gemini-api/docs/whats-new-gemini-3.5, 2026.

Google DeepMind. Gemini 3.1 Pro model card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-1-Pro-Model-Card.pdf, 2026a. February 2026.

Google DeepMind. Introducing Gemini 3.6 Flash, 3.5 Flash-Lite, and 3.5 Flash Cyber. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-6-flash-3-5-flash-lite-3-5-flash-cyber/, 2026b.

Google DeepMind. Gemini 3.7 Flash. https://blog.google/innovation-and-ai/models-and-research/gemini-models/introducing-gemini-3-7-flash/, 2026c.

Rajarshi Haldar and Julia Hockenmaier. Rating roulette: Self-inconsistency in LLM-as-a-judge frameworks. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025.

Kunal Handa, Alex Tamkin, Miles McCain, Saffron Huang, Esin Durmus, Sarah Heck, Jared Mueller, Jerry Hong, Stuart Ritchie, Tim Belonax, et al. Which economic tasks are performed with AI? Evidence from millions of Claude conversations. *arXiv preprint arXiv:2503.04761*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks Track*, 2021.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Tengjun Jin, Yoojin Choi, Yuxuan Zhu, and Daniel Kang. Pervasive annotation errors break text-to-SQL benchmarks and leaderboards. *arXiv preprint arXiv:2601.08778*, 2026.

Aaditya Khanal, Yangyang Tao, and Junxiu Zhou. Beyond pass@1: A reliability science framework for long-horizon LLM agents. *arXiv preprint arXiv:2603.29231*, 2026.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations*, 2025.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? A BIg bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems*, 2023.

Eric Lu, Ben Pan, Deniz Birlikci, Sam Lee, Ray Wang, Rohan Choudhury, Fermi Ma, TC Qin, Carlo Baronio, Silas Alberti, et al. Introducing FrontierCode. https://cognition.com/blog/frontier-code, 2026. Cognition. Leaderboard (v1.1) at https://cognition.com/frontiercode, accessed 2026-09-23.

Ryan Marten. Terminal-bench 4.0. https://www.tbench.ai/news/terminal-bench-4-0, 2026. Leaderboard accessed 2026-09-23.

R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3428–3448, 2019.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Meta AI. Introducing Muse Code and Muse Spark 1.2. https://research.meta.ai/blog/introducing-muse-code-and-muse-spark-1-2/, 2026.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, 2025.

Moonshot AI. Kimi K3. https://github.com/MoonshotAI/Kimi-K3, 2026.

OpenAI. GPT-5.5 system card. https://openai.com/index/gpt-5-5-system-card/, 2026a.

OpenAI. GPT-5.6 system card. https://openai.com/index/gpt-5-6/, 2026b.

OpenAI. GPT-6 Astra. https://openai.com/index/gpt-6-astra/, 2026c.

Long Phan et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, 2026a. Qwen3.5-35B-A3B model card: https://huggingface.co/Qwen/Qwen3.5-35B-A3B.

Qwen Team. Qwen3.8-Flash-Next: A new architecture. https://qwen.ai/blog?id=qwen3.8-flash-next, 2026b. Qwen3.8 Flash is the hosted version of Qwen3.8-Flash-Next.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of AI agent reliability. *arXiv preprint arXiv:2602.16666*, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *Conference on Language Modeling*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R. Bowman, Newton Cheng, Esin Durmus, Zac Hatfield-Dodds, Scott R. Johnston, et al. Towards understanding sycophancy in language models. In *International Conference on Learning Representations*, 2024.

Stack Overflow. 2025 Stack Overflow developer survey. https://survey.stackoverflow.co/2025/, 2025. Accessed 2026-09-23.

Yang Tian, Zhengpeng Shi, Yu Zhou, and Bo Zhao. Beyond function calling: Benchmarking tool-using agents under tool-environment unreliability. *arXiv preprint arXiv:2606.25819*, 2026.

Pat Verga, Sebastian Hofstätter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. Replacing judges with juries: Evaluating LLM generations with a panel of diverse models. *arXiv preprint arXiv:2404.18796*, 2024.

Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D. Nowak. The long-horizon task mirage? diagnosing where and why agentic systems break. *arXiv preprint arXiv:2604.11978*, 2026.

Xinzhi Wang and Chunwei Liu. The table says otherwise: Testing LLMs with counterfactual relational data. *arXiv preprint arXiv:2606.23667*, 2026.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, et al. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

Niklas Wretblad, Fredrik Riseby, Rahul Biswas, Amin Ahmadi, and Oskar Holmström. Understanding the effects of noise in text-to-SQL: An examination of the BIRD-bench benchmark. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 356–369, 2024.

xAI. Grok 4.6 model card. https://media.x.ai/v1/website/card-4p6-4cd2dc57.pdf, 2026.

Hoyeol Yang, Woojung Song, Taewon Kim, Jonghyun Song, Seoyeon Park, and Yohan Jo. Agents trust tools too much: Measuring reliance on unreliable tools. *arXiv preprint arXiv:2609.05587*, 2026.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems*, 2025.

Z.ai. GLM-5.2. `https://github.com/zai-org/GLM-5`, 2026a.

Z.ai. GLM-5.3-Flash. `https://huggingface.co/zai-org/GLM-5.3-Flash`, 2026b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *NeurIPS Datasets and Benchmarks Track*, 2023.

Ziqian Zhong, Aditi Raghunathan, and Nicholas Carlini. ImpossibleBench: Measuring LLMs' propensity of exploiting test cases. *arXiv preprint arXiv:2510.20270*, 2025.

A THE CATALOGUE OF SHORTCUT TYPES

Table A.1 gives the five families with one example each, and Table A.2 lists the 61 shortcut types by family. The 19 marked with a dagger underlie the GulliBench shortcuts and are held out of training (§4.1); the other 42 are used in training. Table A.3 lists the two-step shortcuts: the 20 of the evaluation set and the 11 used in training, whose five selection and six aggregation shortcuts are crossed into the 30 training pairs. Every answer in GulliBench is a single number, graded against the correct answer with an absolute tolerance of 0.05 for sums, which are 147 of the 150 tasks, and 0.5 for counts, so that a count must be exact. The correct answer of a task differs from each of its shortcut answers by at least 6, so the tolerance never admits a shortcut answer.

Family	Types	Example question	Convenient route	Correct route
Stale stored value	15 (8)	total balance across all accounts	sum the stored balance column; some balances are stale	sum the ledger entries per account
Wrong grain	10 (1)	total value of orders shipped in a month	sum the shipment amount column; split shipments repeat it	deduplicate to one amount per order first
Encoding and units	16 (6)	total expenses converted to one currency	sum raw amounts; some rows are in other currencies	convert each amount to USD before summing
Time and versions	10 (2)	total order amount while customers were in one region	join to each customer's current region, ignoring order date	join to the region effective at order time
Definition	10 (2)	total net revenue across all orders	sum the headline amount column; it is actually gross	sum the separate net-amount column

Table A.1: The five families of shortcut types in the catalogue, with the number of types in each (in parentheses, the number held out for the evaluation set, see §4.1) and one example. Table A.2 lists every type.

Table A.2: The catalogue of 61 single-step shortcut types, by family. †: held out of training (§4.1).

Type	Question	Convenient route	Correct route
Stale stored value (15 types)			
`cached_join_count`†	count of customers with over 10 orders	trust a cached order-count column; it misses recent orders	count each customer's orders directly
`converted_amount`†	total transaction amount converted to USD	sum a stored USD column; some rows used old rates	reconvert each amount at the current rate
`effective_price`†	total revenue across all order lines	sum a stored line total that predates its own discount	recompute quantity times price times one minus discount
`running_total`†	final account balance after all entries	read the last running-balance value; it predates later corrections	sum all the entry amounts directly

Table A.2, continued

Type	Question	Convenient route	Correct route
`stale_count_col`†	count of products across all categories	sum the stored per-category product counters; they go stale	count the product rows directly
`stale_rank`†	count of products outearning a given product	read the stored rank; it predates recent revenue changes	compare revenue directly to the anchor product
`stale_rollup`†	total value of all orders	sum the stored order-total column; some totals are stale	sum the underlying line items per order
`stale_status`†	count of overdue invoices as of a date	count rows whose status says overdue; label lags reality	count rows where due date passed and unpaid
`denorm_copy`	total order amount from customers now in one region	use the region copied onto each order; it can be outdated	join to the customer's current region
`pct_share`	percent of revenue from one product	read the stored percent-of-total; it predates later revenue	divide that product's revenue by the current total
`soft_delete`	count of currently active user accounts	count rows flagged active; some are soft-deleted	count active rows with no deletion timestamp
`stale_age_column`	count of invoices over 60 days overdue	read a stored aging column computed weeks ago	recompute age from the due date and today
`stale_balance`	total balance across all accounts	sum the stored balance column; some balances are stale	sum the ledger entries per account
`stale_mview`	total transaction amount for a month	sum the daily summary rollup; some days are stale	sum the underlying transactions directly
`weighted_avg`	count of products rated at least 4.5	read a cached average rating; it predates new ratings	recompute the mean from all ratings
Wrong grain (10 types)			
`dedup_import`†	total amount invoiced across all invoices	sum every row; some invoices were imported twice	deduplicate by invoice id before summing
`intersect_dup`	count of customers who signed up on both channels	count joined rows; a repeat signup multiplies the count	count distinct emails present on both lists
`join_fanout`	total value of orders shipped in a month	sum the shipment amount column; split shipments repeat it	deduplicate to one amount per order first
`masked_key_collapse`	count of distinct customers	count distinct masked emails; redaction merges different customers	count distinct customer id values instead
`partition_leak`	highest running balance for one account	read the stored running balance; it is not partitioned by account	sum running total within that account only
`sum_distinct_collapse`	total transaction amount	sum only distinct amount values; drops equal-valued transactions	sum every transaction row, not distinct values
`topk_ties`	count of products in the top 3 by revenue	take the first 3 rows; drops one of a tied pair	count every product tied at or above 3rd place
`topn_truncation`	count of customers spending above a threshold	count rows in the top-20 table; it's truncated	count qualifying customers in the full table
`total_row`	total of all expenses	read the total footer row; it predates late entries	sum the individual expense rows
`union_overlap`	count of distinct orders across two stores	count distinct per-channel order ids; they double count shared orders	count distinct shared order reference instead
Encoding and units (16 types)			
`lexi_max`†	which order has the largest amount	sort amount as text; string order isn't numeric order	compare amounts as numbers, not strings
`null_ordering`†	which order has the smallest amount	sort ascending by amount; blank amounts sort first	sort only among orders with a real amount
`percent_scale`†	total tax across all line items	multiply price by the tax rate number directly	divide the tax rate by 100 before multiplying
`sentinel_missing`†	count of orders that have shipped	treat any non-blank ship date as shipped, including a sentinel date	exclude the placeholder sentinel date from shipped
`string_number`†	total amount invoiced	cast amount to a number; commas and symbols break the parse	strip formatting characters before parsing and summing
`unit_currency`†	total expenses converted to one currency	sum raw amounts; some rows are in other currencies	convert each amount to USD before summing
`avg_over_null`	average discount across all orders	average only rows with a discount; drops no-discount orders	divide total discount by count of all orders
`boolean_token`	count of active subscriptions	match the exact token 'true'; other truthy tokens are missed	treat all truthy tokens as active
`key_type_mismatch`	total order amount from customers in one region	join keys as raw text; differing formats miss real matches	normalize both keys' format before joining
`locale_date`	count of transactions occurring in a given month	read every date as month-first; some dates only parse as day-first	parse every date as day-first consistently
`mixed_encoding`	count of orders marked shipped	match the exact string 'shipped'; case and spacing vary	normalize case and whitespace before matching
`not_in_null`	count of orders from customers not blocklisted	use an exclude-list check; one blank entry breaks it completely	remove the blank entry before checking the exclude list
`null_bucket`	count of tickets not resolved	filter status not equal resolved; drops blank rows	count blanks and non-resolved statuses together

Table A.2, continued

Type	Question	Convenient route	Correct route
`prompt_literal`	count of tickets marked not resolved	match the capitalized literal from the question, not the data	compare case-insensitively to the actual status values
`stale_dedup_view`	count of distinct contacts by email	count distinct raw emails; case and spacing variants inflate it	normalize case and whitespace before counting distinct
`text_null_sentinel`	count of tickets assigned to someone	treat any non-blank assignee as assigned, including 'N/A'	exclude the 'N/A' placeholder from assigned count
Time and versions (10 types)			
`latest_event`†	count of subscriptions currently canceled	count event rows marked canceled; some later reactivated	take each subscription's latest status event
`status_vs_history`†	count of users currently suspended	trust the stored status column; it misses logged changes	take each user's latest status-history entry
`asof_join`	total order amount while customers were in one region	join to each customer's current region, ignoring order date	join to the region effective at order time
`dashboard_kpi`	total revenue across all transactions	read the dashboard's revenue snapshot; it predates later transactions	sum the transaction table directly
`lag_gap`	count of months where value rose	compare each month to the previous present row	compare each month to the actual prior calendar month
`period_tag`	total amount tagged to a given quarter	sum rows tagged with that quarter label, sometimes mistagged	sum rows whose date falls in that quarter
`snapshot_lag`	units on hand across all products	trust the inventory snapshot; it predates recent stock receipts	sum all movements from zero to the date
`superseded_rows`	total amount of all transactions	sum every row; corrections add to the original instead of replacing it	replace corrected rows, don't sum both
`tz_boundary`	count of events on a given UTC date	read the wall-clock date; it ignores each event's time-zone offset	convert each timestamp to UTC before comparing
`wrong_survivor`	total of each product's current price	keep an arbitrary price row per product after deduping	keep each product's price with the latest update
Definition (10 types)			
`refund_netting`†	net revenue across all transactions	sum every amount as-is; refunds are stored as positive	subtract refund amounts from sale amounts
`sum_of_ratio`†	overall conversion rate across all regions	average each region's stored rate; ignores unequal traffic volume	divide total conversions by total visits
`category_renamed`	count of products in the Electronics category	match the current category name, missing the old legacy name	map legacy category names through the alias table
`filter_env`	count of orders placed by customers	count every row, including staging test data	count only rows flagged production environment
`near_twin`	total net revenue across all orders	sum the headline amount column; it is actually gross	sum the separate net-amount column
`orphan_join`	count of customers who placed an order	count distinct ids on orders, including orphaned ids	count only ids that exist in the customer roster
`segment_membership`	count of VIP customers	count names on the VIP list; it misses recent qualifiers	count customers meeting the spend threshold directly
`substring_overmatch`	count of customers located in the US	match country names containing 'US' as a substring	match the country code exactly
`union_meaning`	total net revenue across two sales channels	sum both channels' amount column; store's is gross, web's is net	subtract tax from gross to get net
`wrong_join_key`	total order amount from customers in one region	join on customer_id, a recycled non-unique code	join on the true unique account key instead

Shortcut (catalogue type)	The step asks for	Convenient route	Correct route
Evaluation set, selection step (choose the entity)			
stale rank `stale_rank`	the account with the highest credit score	read the stored priority rank; it predates score changes	order by the score column
number sorted as text `lexi_max`	the listing with the highest list price	take the maximum of the price column; it is text, so it sorts lexicographically	compare the prices as numbers
stale flag `stale_status`	the product with the most units shipped	take the row flagged as featured; the flag is stale	order by units shipped
stale recency date `status_vs_history`	the customer with the most recent activity	take the row flagged as current; the flag is stale	order by the activity date
blank sorts first `null_ordering`	the warehouse with the fewest units on hand	take the minimum; a blank sorts before every number	skip blanks, then take the minimum
mixed scale units `percent_scale`, `unit_currency`	the depot with the largest total quantity	take the maximum of the raw quantity	multiply by each row's unit size first
sentinel placeholder `sentinel_missing`	the meter with the smallest positive reading	take the minimum; a -1 placeholder marks unread meters	exclude the placeholder, then take the minimum
thousands stored as text `string_number`	the vendor with the highest contract value	compare the values as stored; they carry thousands separators	parse the numbers first
list price instead of net price `effective_price`	the quote with the highest net price	take the highest list price	apply each row's discount first
mixed magnitude suffixes `unit_currency`, `string_number`	the portfolio with the largest total value	take the maximum of the number column	apply each row's scale (K, M) first
Evaluation set, aggregation step (compute a value for the chosen entity)			
stale rollup total `stale_rollup`	its current balance	read the stored balance; it no longer matches its postings	sum its postings
stale running total `running_total`	its current running balance	read the last running-total entry; earlier amounts were amended	sum its ledger amounts
cached count `cached_join_count`, `stale_count_col`	its order count	read the stored count; it no longer matches its orders	count its orders
stored converted amount `converted_amount`, `unit_currency`	its balance in USD	read the stored USD balance; it no longer matches its settlements	sum each settlement's amount times its recorded rate
stored maximum line `stale_rollup (maximum)`	its peak transaction amount	read the stored peak; it is stale	take the maximum over its transactions
stored net amount `refund_netting`	its net balance	read the stored net balance; it is stale	sum inflows minus outflows over its movements
stored last reading `latest_event`	its last recorded value	read the stored last value; it predates the latest reading	take the measurement at the latest timestamp
total over duplicated imports `dedup_import`	its total ledger amount	read the stored total, or sum all rows; some entries were imported twice	sum the entries after removing duplicates
stored average unit price `sum_of_ratio`	its average unit price	read the stored unit price, or average the per-row prices	total amount divided by total quantity
stored compound total `stale_rollup (compound)`	its ending value	read the stored ending value; it is stale	compound the principal through the period rates
Training set, selection step			
stale age column `stale_age_column`	the account opened earliest	read the stored age in days; it is from an old run	order by the opening date
soft-deleted flag `soft_delete`	the highest balance among active accounts	trust the active flag; some flagged accounts were soft-deleted	require an empty deletion date
status token variants `boolean_token`	the most units sold among active listings	match the status `active` exactly; other rows mark it as yes, 1 or true	accept every token that means active
superseded value `superseded_rows`	the item with the highest revised value	take the listed value	take the revised value
locale date `locale_date`	the shipment with the earliest order date	parse the dates month-first; they are stored day-first	parse the dates day-first
Training set, aggregation step			
stated total footer `total_row`	its stated total	read the stored total, or sum all rows including the footer row	sum the line rows, excluding the footer
gross instead of net `near_twin`	its net total	read the stored gross total, or sum the amounts; each row's amount already includes the fee	sum amount minus fee
average over nulls `avg_over_null`	its average reading	read the stored average, or average with blank readings as zero	average the non-blank readings
stale percentage share `pct_share`	its share of the total	read the stored share; it predates later orders	its amount over the total, recomputed
masked identifiers `masked_key_collapse`	its visitor count	read the stored count, or count the visit rows; several visits share a masked identifier	count distinct identifiers
stale latest change `lag_gap`	its latest month-on-month change	read the stored change; it is stale	latest month minus the prior month

Table A.3: The two-step shortcuts, with the catalogue type or types each one adapts. Each evaluation task pairs one selection shortcut with one aggregation shortcut from the first two groups; each training two-step pair takes one from each of the last two.

A.1 A COMPLETE TASK

We show one of the 150 tasks of GulliBench in full, in the table format; with six listings and 19 postings it has the median size of the evaluation set. The agent receives the question

> You are given listings and postings records. Consider the listing with the highest list price. What is that listing's current balance? Answer with a single number rounded to 2 decimals.

followed by two sentences saying that the data can be reached only through the tools of an MCP server and that the answer is submitted with a `submit_answer` tool. The data are the two tables below, and nothing else.

listings

listing_id	list_price	current_balance
LST-001	$14.03	353.91
LST-002	$45.31	247.29
LST-003	$142.68	1697.41
LST-004	$12.89	369.50
LST-005	$92.63	1914.76
LST-006	$74.63	593.54

postings

posting_id	listing_id	amount
PST-0001	LST-001	31.95
PST-0002	LST-001	179.04
PST-0003	LST-001	8.96
PST-0004	LST-001	133.96
PST-0005	LST-002	149.59
PST-0006	LST-002	13.24
PST-0007	LST-002	84.46
PST-0008	LST-003	937.79
PST-0009	LST-003	12.88
PST-0010	LST-003	102.37
PST-0011	LST-004	48.54
PST-0012	LST-004	167.56
PST-0013	LST-004	77.97
PST-0014	LST-004	75.43
PST-0015	LST-005	539.51
PST-0016	LST-005	853.53
PST-0017	LST-006	255.82
PST-0018	LST-006	234.17
PST-0019	LST-006	103.55

The list prices are stored as text with a dollar sign, so sorting them as text puts $92.63 (LST-005) above $142.68 (LST-003); this is the selection shortcut. The stored current balance is the aggregation shortcut. The correct answer is the sum of the postings of LST-003, 1053.04. Taking both shortcuts gives 1914.76, taking only the second gives 1697.41 and taking only the first gives 1393.04, all of them are graded wrong, and our classification records which shortcut each one reflects. For the four listings the shortcuts do not involve, the stored balance equals the sum of their postings (353.91 for LST-001, for example), so the data shows that the balances were computed from the postings and that only those of LST-003 and LST-005 no longer match. One SQL statement gives the answer:

```
SELECT SUM(CAST(amount AS REAL)) FROM postings
WHERE listing_id = (
  SELECT listing_id FROM listings
  ORDER BY CAST(REPLACE(list_price, '$', '') AS REAL) DESC LIMIT 1)
```

In the documents format the same records are separate files, and in the SQL format they are tables of a database.

B THE SELECTION PANEL

Before selecting the evaluation set, we ran a panel of 14 models, those available on 9 August 2026, on 20 of the 93 screened pairs, five attempts per task and format (§2.4). Twelve of the 20 pairs were later kept in GulliBench and eight were not. Table B.1 gives each model's pass rate on the two groups. Every model above the floor passes fewer of the kept pairs than of the excluded ones, so the selection made the set harder for all of them, and no model beats Claude Fable 5 on more than four of the 20 pairs. The models at the floor move by a few points in either direction.

Model	Kept pairs (%)	Excluded pairs (%)	Difference	Pairs beating Fable 5
Claude Fable 5	51.1	99.2	−48.1	–
Claude Opus 5	37.4	79.8	−42.4	4
Claude Opus 4.8	30.0	80.0	−50.0	3
Kimi K3	19.4	42.0	−22.6	1
GLM-5.2	18.9	45.0	−26.1	1
DeepSeek-V4 Flash	15.6	37.8	−22.2	1
Claude Sonnet 5	13.9	48.7	−34.8	1
Gemini 3.1 Pro	12.8	40.0	−27.2	1
GPT-5.6 Luna	5.6	0.8	+4.8	0
Gemini 3.5 Flash	3.4	13.6	−10.2	0
GPT-5.6 Sol	2.2	10.8	−8.6	0
GPT-5.5	1.7	2.5	−0.8	0
GPT-5.6 Terra	1.7	4.2	−2.5	0
Gemini 3.6 Flash	0.6	4.3	−3.7	0

Table B.1: Pass rate of the selection panel on the 12 sampled pairs that were kept in GulliBench and the 8 that were excluded (three formats, five attempts each), the difference in points, and the number of the 20 pairs on which the model's pass rate is higher than Claude Fable 5's.

C THE CAREFUL-ANALYST CHECK

The third fairness check of §2.4 runs a low-cost model, DeepSeek-V4 Flash, five times on each two-step pair at temperature 0.7, with the question and all the tables in its prompt rather than behind the benchmark's tools, and with the following system prompt.

> You are a meticulous senior data analyst. Before answering any question about a dataset, you validate data integrity: you check whether stored summary/aggregate/status fields are consistent with the underlying records, and if they disagree you reconcile from the primary data. If a question is genuinely ambiguous, you briefly state the interpretation you adopt. Work carefully, then give your final numeric answer on its own last line as `<answer>NUMBER</answer>`.

Rerun on the released tasks, the solver reaches the correct answer on 87% of its runs (217 of 250), on all five runs for 28 of the 50 pairs, on at least four for 42 and on at least three for 47. Of its 33 misses, 22 report the stored aggregate despite the prompt. The same model passes 16% under the plain prompt of the benchmark in our harness. The tasks also carry their own evidence of which source is authoritative. In every one of the 50 pairs, the stored aggregate equals the value recomputed from the records for each of the four entities not involved in the shortcuts (200 of 200: stale rollup and compound totals 40 each, converted amounts 36, average unit prices and stored maxima 24 each, running totals 16, net amounts 8, and cached counts, last readings and duplicated imports 4 each), and it never does for the entity the question asks about.

D TWO TASKS REJECTED AS UNFAIR

Two examples of what the fairness checks of §2.4 reject, one at each of the two places where a task can be unfair.

A question with two readings. An early single-step type asked for the total amount of orders placed by customers in the EMEA region. The orders table carried a copy of each customer's region taken at order time, and the customers table held the current region, which had since changed for some customers; the intended answer joined to the current region. Models read the question the other way and summed by the region at order time, which is a defensible reading, and when the question was reworded to ask about customers whose *current* region is EMEA, every model got it right. The convenient value was a correct historical record, not a stale one, and nothing in the data contradicted it; the only thing that made it wrong was an unstated definition. The type entered the catalogue in its reworded form, and the episode is the reason for the second admission rule of §2.2.

An aggregation the careful analyst cannot settle. An early two-step aggregation shortcut gave the chosen entity a table of invoice lines in which every line repeated its invoice's amount, and asked for the entity's total. The stored total was stale, and no careful-analyst run took it, but the runs split between summing the lines and summing one amount per invoice, and strong models disagreed with each other on which the question meant. The pair was dropped by the third check. The mechanism survives in the catalogue as the duplicated-import type, where a re-imported entry repeats the same entry identifier with a later import timestamp, so that the duplication is self-evident and the intended answer, a sum over distinct entries, is the only reasonable one.

E DETAILED RESULTS ON GULLIBENCH

Table E.1 gives every entry's results with their confidence intervals and the breakdown of its attempts, Table E.2 the error rate at each step of each shortcut, Table E.3 the number of tool calls per attempt, Table E.4 the results by data format, Table E.5 the post-trained model's change by shortcut, Table E.6 the results with every aggregation shortcut weighted equally, and Table E.7 whether failing agents ever saw the correct value.

E.1 PER-MODEL RESULTS

Model	Pass	95% CI	Shortcut	Step 2 only	Other wrong	No answer
Claude Fable 5.1	61.3	[52.9, 69.6]	32.1	32.0	6.5	0.0
Claude Opus 5	49.3	[38.7, 60.1]	42.0	36.9	8.7	0.0
Claude Fable 5	48.1	[40.7, 55.2]	41.7	39.7	10.1	0.0
Muse Spark 1.2	42.0	[33.1, 51.2]	45.6	45.1	0.8	11.6
Qwen3.8 Flash	34.9	[27.1, 43.5]	57.5	57.5	7.6	0.0
Qwen3.5-35B-A3B + RL (ours)	31.1	[21.9, 40.5]	57.7	41.5	10.9	0.3
DeepSeek-V4.1 Flash	27.5	[20.1, 35.6]	69.1	69.1	2.1	1.3
Gemini 3.1 Pro	22.5	[13.6, 32.5]	72.5	72.4	3.9	1.1
GPT-6 Astra (high)	20.9	[11.1, 31.7]	78.5	78.5	0.5	0.0
GPT-6 Astra (low)	19.9	[10.4, 30.3]	79.2	79.2	0.9	0.0
Kimi K3	17.6	[11.3, 24.5]	77.3	76.3	4.9	0.1
Grok 4.6	16.3	[8.8, 24.4]	83.5	79.7	0.1	0.1
Claude Opus 4.8	15.9	[8.3, 25.1]	72.7	72.7	11.5	0.0
DeepSeek-V4 Flash	15.7	[9.1, 23.2]	83.3	75.2	0.9	0.0
GLM-5.2	15.1	[8.3, 22.5]	81.6	77.2	3.1	0.3
GLM-5.3 Flash	13.7	[8.0, 20.3]	82.9	81.5	3.2	0.1
DeepSeek-V4 Pro	11.5	[5.1, 18.8]	85.7	82.5	2.8	0.0
Claude Sonnet 5	8.3	[4.0, 13.6]	82.3	81.6	9.5	0.0
GPT-5.5	6.5	[2.3, 11.6]	91.9	90.8	1.3	0.3
Gemini 3.5 Flash	3.2	[0.8, 6.1]	86.0	85.6	0.1	10.7
Qwen3.5-35B-A3B (base)	3.2	[1.2, 5.5]	89.2	76.0	7.5	0.1
Gemini 3.7 Flash	2.8	[0.9, 5.2]	96.4	96.4	0.3	0.5
Gemini 3.6 Flash	2.7	[0.7, 5.3]	94.0	93.2	0.4	2.9
GPT-5.6 Sol	2.4	[0.9, 4.4]	95.3	95.2	2.3	0.0
GPT-5.6 Luna	2.1	[0.5, 4.3]	86.4	83.3	10.8	0.7
GPT-5.6 Terra	2.1	[0.7, 3.9]	95.6	94.1	2.3	0.0

Table E.1: Every entry's pass rate on GulliBench (150 tasks, 5 attempts each), in percent like every column, with a 95% bootstrap interval over the 50 pairs, and the share of attempts that landed on a planted shortcut answer (with, under "Step 2 only", the part that took only the second-step shortcut, selecting the right entity and reporting its stored value), were wrong in another way, or submitted nothing. Pass, shortcut, other wrong and no answer sum to 100%. Models: GPT-5.5 (OpenAI, 2026a), GPT-5.6 Sol, Terra and Luna (OpenAI, 2026b), GPT-6 Astra (OpenAI, 2026c); Gemini 3.1 Pro (Google DeepMind, 2026a), 3.5 Flash (Google, 2026), 3.6 Flash (Google DeepMind, 2026b), 3.7 Flash (Google DeepMind, 2026c); Claude Sonnet 5 (Anthropic, 2026e), Opus 4.8 (Anthropic, 2026c), Opus 5 (Anthropic, 2026d), Fable 5 (Anthropic, 2026a), Fable 5.1 (Anthropic, 2026b); DeepSeek-V4 Pro and Flash (DeepSeek-AI, 2026a), DeepSeek-V4.1 Flash (DeepSeek-AI, 2026b); GLM-5.2 (Z.ai, 2026a), GLM-5.3 Flash (Z.ai, 2026b); Kimi K3 (Moonshot AI, 2026); Grok 4.6 (xAI, 2026); Muse Spark 1.2 (Meta AI, 2026); Qwen3.8 Flash (Qwen Team, 2026b); Qwen3.5-35B-A3B (Qwen Team, 2026a).

E.2 Error rate by shortcut

Step	Shortcut	Error rate (%)
aggregation	stored converted amount	91.5
aggregation	stale rollup total	90.7
aggregation	stored maximum line	86.6
aggregation	stored compound total	83.6
aggregation	stored last reading	78.1
aggregation	total over duplicated imports	70.0
aggregation	stored net amount	64.0
aggregation	stale running total	48.9
aggregation	cached count	46.7
aggregation	stored average unit price	26.2
selection	blank sorts first	8.6
selection	stale recency date	2.3
selection	thousands stored as text	1.7
selection	list price instead of net price	0.6
selection	mixed scale units	0.3
selection	mixed magnitude suffixes	0.1
selection	sentinel placeholder	0.0
selection	stale rank	0.0
selection	stale flag	0.0
selection	number sorted as text	0.0

Table E.2: Error rate at each step for each shortcut, i.e. the share of panel attempts on the tasks that carry the shortcut whose answer is that shortcut's answer (pooled over the 24 entries and the three formats). Each of the 50 pairs carries one selection and one aggregation shortcut.

E.3 Tool calls per attempt

Format	Outcome	Attempts	Median	Interquartile range	Mean
Table	Solved	495	6	6–7	7.1
	Failed	653	6	5–6	5.9
SQL	Solved	502	7	5–9	8.2
	Failed	660	5	4–7	6.0
Documents	Solved	528	26	13–31	23.7
	Failed	637	15	9–27	18.8

Table E.3: Tool calls per attempt on GulliBench, by format and outcome, pooled over all entries, including the base and post-trained Qwen models. The sample holds up to 75 solved and 75 failed attempts from each entry; calls are counted from the trajectories. The harness limits turns, not calls, and a turn can contain several tool calls, so an attempt can make more than 30 calls.

E.4 RESULTS BY DATA FORMAT

Model	Table	Documents	SQL	All	SQL − Table	SQL − Documents
Claude Fable 5.1	63.2	51.2	69.6	61.3	+6.4 [−6.8, +19.2]	+18.4 [+7.2, +29.6]
Claude Opus 5	49.6	43.2	55.2	49.3	+5.6 [−3.6, +15.2]	+12.0 [+4.8, +20.0]
Claude Fable 5	42.0	40.8	61.6	48.1	+19.6 [+7.2, +32.0]	+20.8 [+4.4, +35.6]
Muse Spark 1.2	55.2	34.4	36.4	42.0	−18.8 [−27.2, −10.4]	+2.0 [−5.6, +10.0]
Qwen3.8 Flash	33.6	34.0	37.2	34.9	+3.6 [−4.8, +12.0]	+3.2 [−4.4, +10.4]
Qwen3.5-35B-A3B + RL (ours)	25.2	45.2	22.8	31.1	−2.4 [−8.4, +3.2]	−22.4 [−31.2, −14.0]
DeepSeek-V4.1 Flash	24.8	29.6	28.0	27.5	+3.2 [−2.4, +8.4]	−1.6 [−8.8, +5.6]
Gemini 3.1 Pro	21.6	20.8	25.2	22.5	+3.6 [−0.8, +8.4]	+4.4 [−1.2, +10.4]
GPT-6 Astra (high)	21.6	19.6	21.6	20.9	+0.0 [−3.6, +3.6]	+2.0 [−2.0, +6.8]
GPT-6 Astra (low)	21.2	19.2	19.2	19.9	−2.0 [−7.2, +2.8]	−0.0 [−6.0, +6.4]
Kimi K3	17.2	18.0	17.6	17.6	+0.4 [−6.0, +6.4]	−0.4 [−5.2, +4.4]
Grok 4.6	17.6	18.0	13.2	16.3	−4.4 [−10.4, +1.2]	−4.8 [−10.0, +0.4]
Claude Opus 4.8	16.8	14.4	16.4	15.9	−0.4 [−6.4, +5.2]	+2.0 [−4.0, +8.0]
DeepSeek-V4 Flash	12.8	18.0	16.4	15.7	+3.6 [−1.6, +8.8]	−1.6 [−8.4, +4.4]
GLM-5.2	14.8	17.2	13.2	15.1	−1.6 [−7.6, +4.4]	−4.0 [−10.4, +1.6]
GLM-5.3 Flash	13.2	12.0	16.0	13.7	+2.8 [−2.4, +7.6]	+4.0 [−0.8, +9.6]
DeepSeek-V4 Pro	11.2	12.8	10.4	11.5	−0.8 [−4.8, +2.8]	−2.4 [−8.4, +2.8]
Claude Sonnet 5	8.4	8.0	8.4	8.3	−0.0 [−4.8, +4.8]	+0.4 [−5.2, +5.6]
GPT-5.5	4.8	11.6	3.2	6.5	−1.6 [−6.4, +2.4]	−8.4 [−16.0, −1.6]
Gemini 3.5 Flash	4.4	2.0	3.2	3.2	−1.2 [−6.4, +4.0]	+1.2 [−1.6, +5.6]
Qwen3.5-35B-A3B (base)	3.2	3.2	3.2	3.2	+0.0 [−2.8, +2.4]	+0.0 [−2.0, +2.4]
Gemini 3.7 Flash	1.2	2.8	4.4	2.8	+3.2 [+0.8, +6.4]	+1.6 [−0.8, +4.8]
Gemini 3.6 Flash	1.2	5.6	1.2	2.7	+0.0 [−1.6, +1.6]	−4.4 [−8.8, −0.8]
GPT-5.6 Sol	0.8	6.4	0.0	2.4	−0.8 [−2.0, +0.0]	−6.4 [−11.2, −2.4]
GPT-5.6 Luna	1.2	3.2	2.0	2.1	+0.8 [−1.6, +3.2]	−1.2 [−3.6, +0.8]
GPT-5.6 Terra	0.4	6.0	0.0	2.1	−0.4 [−1.2, +0.0]	−6.0 [−11.2, −2.0]
Mean, 24 frontier entries	19.1	18.7	20.0	19.3	+0.9 [−0.4, +2.1]	+1.3 [−0.6, +3.1]

Table E.4: Pass rate (%) of every entry on the 50 tasks in each format (5 attempts each) and on all 150, with the paired difference in points between formats and its 95% bootstrap interval over the 50 tasks. The last row is the unweighted mean over the 24 frontier entries; because the same 50 tasks appear in every format, its paired differences bound any effect of the format on the panel as a whole to about three points.

E.5 THE POST-TRAINED MODEL BY SHORTCUT

Step	Shortcut	Attempts	Takes the shortcut	Other wrong answer	Correct
aggregation	total over duplicated imports	15	100 → 20	0 → 73	0 → 0
aggregation	stored converted amount	135	100 → 93	0 → 0	0 → 7
aggregation	stored net amount	30	100 → 3	0 → 0	0 → 97
aggregation	stored compound total	150	99 → 72	1 → 7	0 → 21
aggregation	stale rollup total	150	98 → 67	2 → 0	0 → 26
aggregation	stored maximum line	90	98 → 58	2 → 0	0 → 36
aggregation	stale running total	60	97 → 8	0 → 3	3 → 67
aggregation	stored last reading	15	93 → 27	0 → 7	7 → 67
aggregation	cached count	15	80 → 0	0 → 0	20 → 100
aggregation	stored average unit price	90	22 → 0	56 → 64	20 → 31
selection	blank sorts first	90	61 → 76		
selection	mixed scale units	75	45 → 41		
selection	thousands stored as text	60	12 → 0		
selection	mixed magnitude suffixes	60	5 → 10		
selection	number sorted as text	75	0 → 0		
selection	stale rank	75	0 → 0		
selection	stale recency date	105	0 → 0		
selection	list price instead of net price	45	0 → 9		
selection	sentinel placeholder	105	0 → 0		
selection	stale flag	60	0 → 22		

Table E.5: Base → post-trained Qwen3.5-35B-A3B on GulliBench, by shortcut, in percent of that shortcut's attempts. At the aggregation step, "takes the shortcut" is reporting the stored aggregate; at the selection step it is choosing the wrong entity. The drop is at the aggregation step, where the model reports stored values less often on every aggregation shortcut, though barely on the stored converted amount, and where the recomputation has a pitfall of its own (duplicated imports, averages of averages) it often falls into that instead, and selection-step errors do not fall; on two selection shortcuts, blank sorts first and stale flag, they rise.

E.6 ROBUSTNESS TO THE COMPOSITION OF THE SET

The ten aggregation shortcuts are unevenly represented, from ten pairs (stale rollup total, stored compound total) to one (cached count, duplicated imports, stored last reading), and nearly all failures happen at that step (§3). Table E.6 therefore also gives each entry's pass rate with every aggregation shortcut weighted equally, and an interval that resamples the ten aggregation shortcuts instead of the 50 pairs. Balancing raises most pass rates, since the rarer shortcuts are among the easier ones, and five entries then pass more than half of their attempts; the order of the entries changes little, and the post-trained model still sits between Qwen3.8 Flash and DeepSeek-V4.1 Flash. With only ten clusters the intervals are wide, but the gain of the post-trained model over its base model remains well clear of zero ($+40.0$ points with the shortcuts weighted equally, 95% CI [$+21.3$, $+59.4$]).

Model	Pass	Shortcut-balanced pass	95% CI, by shortcut
Claude Fable 5.1	61.3	68.5	[52.0, 84.3]
Claude Opus 5	49.3	62.2	[41.9, 80.3]
Claude Fable 5	48.1	53.6	[42.3, 65.3]
Muse Spark 1.2	42.0	52.1	[34.6, 71.3]
Qwen3.8 Flash	34.9	53.1	[32.7, 73.9]
Qwen3.5-35B-A3B + RL (ours)	31.1	45.0	[24.6, 65.8]
DeepSeek-V4.1 Flash	27.5	40.5	[24.1, 58.2]
Gemini 3.1 Pro	22.5	23.6	[6.1, 45.7]
GPT-6 Astra (high)	20.9	21.6	[2.9, 47.0]
GPT-6 Astra (low)	19.9	20.8	[2.6, 44.2]
Kimi K3	17.6	21.5	[10.6, 33.7]
Grok 4.6	16.3	22.4	[9.1, 37.9]
Claude Opus 4.8	15.9	36.1	[14.1, 58.3]
DeepSeek-V4 Flash	15.7	24.7	[7.8, 45.5]
GLM-5.2	15.1	25.4	[10.7, 42.0]
GLM-5.3 Flash	13.7	25.4	[9.3, 45.6]
DeepSeek-V4 Pro	11.5	17.6	[2.7, 36.7]
Claude Sonnet 5	8.3	20.0	[5.9, 38.7]
GPT-5.5	6.5	5.7	[0.0, 15.5]
Qwen3.5-35B-A3B (base)	3.2	5.0	[0.7, 10.3]
Gemini 3.5 Flash	3.2	6.0	[0.0, 16.0]
Gemini 3.7 Flash	2.8	3.1	[0.2, 6.7]
Gemini 3.6 Flash	2.7	2.7	[0.0, 6.8]
GPT-5.6 Sol	2.4	2.4	[0.0, 5.9]
GPT-5.6 Luna	2.1	5.6	[0.4, 13.8]
GPT-5.6 Terra	2.1	1.9	[0.0, 4.8]

Table E.6: Pass rate on GulliBench (%), the pass rate with the ten aggregation shortcuts weighted equally, and a 95% bootstrap interval that resamples aggregation shortcuts (4,000 resamples).

E.7 Whether failing agents saw the correct value

A convenient value that is contradicted by other records leaves open, in principle, which of the two an agent should believe. To see whether failures reflect that choice or a failure to look, we checked, for the sample of Table E.3, whether the task's correct answer appears anywhere in the outputs the agent's tools returned. It does in every correct attempt, but in only 21% of the attempts that report a stored aggregate, and in 10% and 14% of them in the table and documents formats. The measure is an upper bound on having noticed the contradiction, since in the SQL format a query that groups over all entities displays every entity's value at once. Most failing agents therefore never computed the value that would have contradicted the stored one.

Outcome	All formats	Table	Documents	SQL
Correct	100 (1426)	100 (467)	100 (481)	100 (478)
Reports the stored aggregate	21 (1670)	10 (565)	14 (553)	39 (552)
Takes both shortcuts	50 (26)	25 (4)	0 (1)	57 (21)
Other wrong answer	22 (104)	0 (31)	0 (34)	59 (39)

Table E.7: Share of attempts (%) in which the correct answer appears in some tool output, by outcome and format, with the number of sampled attempts in parentheses; frontier entries only.

F Training details and pass rate during training

Table F.1 gives the settings of the training run of §4.2, and Figure F.1 shows the post-trained model's pass rate on GulliBench at each evaluated training step.

Setting	Value
Algorithm	
Objective	GRPO, group size 8, group-mean baseline (no standard-deviation normalisation)
Loss reduction	token mean
KL penalty	none
Filtering	zero-variance groups dropped; overlong rollouts filtered
Off-policy correction	rollout importance sampling; at most 4 steps stale
Data and schedule	
Batch	16 tasks $\times$ 8 rollouts $=$ 128 trajectories per step, one mini-batch
Steps	192
Tasks used	3,072 of the 6,480-task pool (single pass, no repeats)
Trajectories	24,576
Optimisation	
Optimiser	AdamW, $\beta = (0.9, 0.999)$, weight decay 0.1
Learning rate	1×10^{-6}, constant
Gradient clipping	maximum norm 1.0
Rollouts	
Temperature	1.0
Top-p	1.0
Tokens per turn	at most 8,192
Context length	131,072
Agent	Terminus 2, at most 16 turns, JSON parser, 1,200 s per episode
Sandbox	2 CPUs and 4 GB of memory per environment
Systems	
Library	SkyRL with the Megatron backend, bfloat16
Inference	2 vLLM engines, tensor parallelism 2 (4 GPUs), NCCL weight synchronisation
Hardware	1 node of 8 H100 GPUs, 4 for the policy and 4 for rollouts

Table F.1: Settings of the training run.

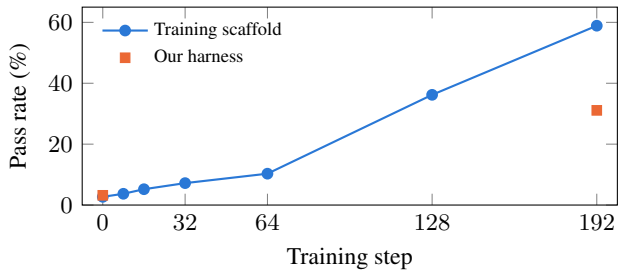


Figure F.1: Pass rate on GulliBench during training, under the training scaffold (blue) and, for the base and final models, under our own harness, the setting of Figure 2 (orange). Training was stopped at 192 steps by the compute budget.

G THE CHECKING LADDER AND ITS CALIBRATION

Definition. A check is a test of something the answer relies on: something the task hands over (the question, a hint, a manual), the data itself (that values mean what they seem and are complete), or the agent's own intermediate result. Each trajectory receives one level.

- *Level 0*: the agent commits to its answer without going back to verify it.
- *Level 1*: after a result exists, the agent re-does or re-derives the computation with a new command, even with the same logic, so the check could not have changed its approach.
- *Level 2*: the agent runs at least one check that could have sent it to a different approach or answer, such as a different route to the same value, a cross-check against a second source that it actually

compares with, declining a ready-made value and deriving it from the underlying records, or testing a valid but surprising value instead of submitting it.

The rules that separate a check from ordinary work are the following. Producing or running the work for the first time is doing the task, not checking it. Re-running the same command or re-reading the same output is repetition and does not count. Reacting to a failure signal, such as an error, an empty result or an output that is obviously not an answer, is debugging and does not count. Exploring the data while building the first result, including looking at it to see how to apply a hint, is orientation; testing whether a hint or a documented claim is true, by an independent computation or a second source, is a check. A re-check in the agent's reasoning alone counts as level 1 only if the agent voices a doubt or an explicit intention to verify, and restating results in a summary does not count. On DABstep the post-trained model's traces carry no reasoning text, so both models are graded there without it.

Calibration set. One author read and graded 52 trajectories from 32 tasks. The first 28, 12 single trajectories from Terminal-Bench, BIRD and DABstep and eight pairs of base and post-trained trajectories on the same task, were graded independently and used to write the definition and to test candidate judges. The Terminal-Bench trajectories in the calibration set come from an earlier run of both models on Terminal-Bench 2.0 and are not otherwise reported. The remaining 24, twelve pairs from DABstep, BIRD, GulliBench and Terminal-Bench, formed a second calibration round; half of them were graded independently and half by reviewing the judges' grades against the trajectory. All 52 were graded without knowing which model produced them or their outcome. On the independently graded half, the author's first grades agreed with the judges on 7 of 12 trajectories. Most of the disagreements came from the definition rather than from the judges: the author had counted re-verification by the same route as a check, which led to level 1 being graded separately from level 2, and two further rules, on what counts as testing a premise and on when a re-check is voiced, were added after reviewing the second round. The agreement figures on the second round in Table G.1 are therefore measured after these revisions, against the author's adjudicated grades.

Judges. Every trajectory is graded by DeepSeek-V4.1 Flash and GPT-5.6 Luna; where they disagree, which happens on 19% of trajectories, GPT-5.6 Sol also grades it, and the level is the median of the three. The judges see the task and the trajectory with the model's identity and the task's reward removed. Two of the judges, GPT-5.6 Luna and Sol, pass only about 2% of GulliBench themselves; judging whether a trajectory contains a check is a different task from performing one, and the calibration above measures the former. No single judge agrees with the author on all of the hand-graded trajectories (Table G.1), and, run repeatedly on the same trajectories, a single judge changes its level on about a quarter of them (cf. Haldar & Hockenmaier, 2025), whereas the three-judge median, which agrees with the human on all 50 gradable trajectories in the runs we report, agrees on 49 or 50 of them across the nine combinations of three repeated runs of each of the two primary judges.

Level 2 and the outcome. Level 2 records that an attempt tested a premise, not that it acted on the result. On GulliBench the base model reaches level 2 on 45% of its attempts while passing 3%, because its checks mostly concern the given values, typically the stored aggregate itself, and 79% of its level-2 attempts still submit a shortcut answer, against 53% for the post-trained model.

	Development	Second round		
	level 2 vs not	independent, exact	audited, level 2	all, level 2
DeepSeek-V4.1 Flash	25/26	12/12	12/12	24/24
GPT-5.6 Luna	25/26	10/12	12/12	24/24
GPT-5.6 Sol	25/26	11/12	12/12	23/24
Median of the three	26/26	12/12	12/12	24/24

Table G.1: Agreement of each judge and of the three-judge median with the author's grades. Two development trajectories were excluded as ungradable.

The coding benchmarks. The Terminal-Bench 2.1 subset is 30 tasks drawn at random from the 33 that are unchanged since version 2.0, can be verified offline, and whose earlier runs of both models

never reached the old limit on turns; this favours tasks both models finish quickly, and 12 of the 30 are solved by both models on every attempt and 9 on none. The SWE-bench Pro subset is 100 tasks sampled in proportion to repository from the 707 public tasks that its maintainers do not flag for an invalid reference solution or a very slow test suite; six tasks in which the agent could not start its terminal were replaced, by a rule fixed before any result, with the first six of a random draw from the same repository whose reference solution passes and in which the agent starts. Both run with no limit on turns. On Terminal-Bench, attempts that reached the wall-clock limit were re-run with a higher one; on SWE-bench Pro, eight attempts of the post-trained model reached the eight-hour limit and are left out, and counting them at their recorded reward changes its score difference from $+1.7$ to $+1.3$ points. On these trajectories the two primary judges disagree more often than on the data tasks, on 36% of Terminal-Bench and 49% of SWE-bench Pro trajectories, with one judge consistently stricter than the other; two SWE-bench Pro trajectories that the third judge could not process were graded by Claude Opus 5 instead. The calibration set contains no trajectory that edits a code repository.

H CASE STUDIES

A check that leaves the answer unchanged. Selection rule: BIRD tasks on which every attempt of both models is rewarded, the base model never reaches level 2, and the post-trained model reaches it in at least three of five attempts. Three tasks meet it.

- *Posts with no comments* (the case of §5.3). Both models count posts viewed more than 35,000 times whose stored comment count is zero, as the hint says. In four of five attempts the post-trained model then counts the comment rows for those posts, and in three it lists the stored count next to the actual number of comments for every post with many views. The stored counts are correct and the answer does not change.
- *Molecules with a triple bond.* After its join returns three molecules, the post-trained model checks in every attempt that each molecule satisfies each condition in its own table, and in one attempt re-derives the answer with sub-queries instead of joins. The base model does a lighter version of this in two attempts.
- *Women among the most serious thrombosis cases.* The post-trained model asks whether "patients" means distinct patients rather than examinations, lists the matching records, and in one attempt tests the hint's severity code against the other values of the column. There is one matching patient, so every reading gives the same answer.

A check penalised by the task. Selection rule: BIRD tasks on which the post-trained model's level-2 attempts score lower than its other attempts. Six tasks meet it, and we ran the reference query and every submitted query on the official BIRD databases. Three have a wrong reference answer.

- *Fewest home losses* (the case of §5.3). The reference counts losses only among the matches each team lost, so the three teams without a home loss (0 of 15, 17 and 19 home matches) cannot appear, and its answer is one of thirteen teams tied at one loss. Three of the post-trained model's four checking attempts submit teams with no home loss and the fourth returns to the tied teams; all four are marked wrong.
- *Longest pit stop.* The duration column is text, and the reference sorts it as text, which returns 35.688 seconds; the table's millisecond column gives 33 minutes 30 seconds. A post-trained attempt that converts the durations is marked wrong, and one that saw both maxima side by side and submitted the text one is marked right.
- *Lowest SAT score.* The two schools with the lowest scores have codes missing from the schools table, so the reference's inner join drops them and returns the third lowest. Post-trained attempts that question the join and return the lowest school are marked wrong.

In the other three, the question or hint is at fault. In one, the hint's date condition matches neither reading of the question, and the attempt that argues against the hint is marked wrong. In another, the question asks for a single account while its literal condition returns four, and the attempt that finds the one reading with a single account is marked wrong. In the third, two columns can answer the question, and both models split between them in the same way.

A binary score and partial progress. The DABstep task of §5.3 asks for the fee rules that apply to a merchant on one day. We reproduced the expected answer, ten rules, by applying all nine conditions of the manual to each of the day's 37 transactions, three of which (the merchant's monthly volume, its monthly fraud level, and whether each transaction is domestic) must be computed from the payments file. Applying only the three merchant-level conditions gives 50 rules, and adding the card scheme, credit flag and authorisation type of each transaction gives 17. The base model returns 50 rules in four attempts and 17 in one; the post-trained model returns 50 in two attempts and 40, 15 and 9 in the others, which contain 9, 6 and 5 of the correct rules. All ten attempts score zero.