# Separating Reading from Execution in Counterfactual Question Answering

Anonymous authors
Paper under double-blind review

## Abstract

A counterfactual question over an explicit causal story asks what would have happened had one variable been different. A language model answering in one pass does two jobs at once: it reads the story into a causal model and it computes the answer. We separate them: a schema-constrained call extracts a boolean causal model, and a deterministic executor enumerates every world that model allows and answers exactly. With a 14B model, this pipeline matches the best published CounterBench result, obtained with a 671B model (91.6% against 91.8%), and beats the strongest prompt we built at the same size. We then use the separation to ask when exact execution is worth its cost. Handed the same stored model the executor receives, a tuned prompt is not distinguishable from the executor where the question already fixes the story's hidden causes (95.6% against 93.1% on 496 items). The advantage appears where answering requires abduction, inferring unobserved causes before intervening: on 325 pre-registered abduction items the executor scores 96.0% against 74.8%, 21.2 points (95% interval $[+16.3, +26.2]$). The gap holds for a second model family, a larger model and majority voting, at every number of hidden causes (up to ten) and depth (up to nine rules) we tested, and most of it disappears when the hidden causes are disclosed; a prompt built specifically for abduction did worse. A reasoning model does close the gap on the same extracted problems (gpt-oss-120b, 95.4%), at a median of 64 seconds and 1,376 generated tokens per problem, where the executor needs under a millisecond. The executor also audits the benchmark: it finds 60 of CounterBench's official answers inconsistent with their own stories, each with a hand-checkable certificate. On CLadder, whose template-generated stories use everyday wording that is harder to read, a scaffolded prompt edges two points ahead of the pipeline.

## 1 Introduction

A counterfactual question asks what would have happened under a change that did not occur. Take the story "The alarm rings if the husband or the wife sets it. The alarm rang." and the question "Would the alarm have rung if the husband had not set it?". Benchmarks such as CLadder (Jin et al., 2023) and CounterBench (Chen et al., 2026) pose thousands of such questions over short stories that state every causal rule, often with nonsense variable names so that world knowledge cannot help. Language models do poorly on them, and their accuracy falls as the causal chains get longer.

Answering takes two steps. Reading turns the sentences into a causal model: its variables, rules, observations and the hypothetical change. Computing works out the answer from that model. Prior work finds that models recover causal structure much better than they use it (Gendron et al., 2024; Yamin et al., 2024), which points at the computation. The computation can also be done exactly by a short program. We build a pipeline that does so: a schema-constrained reader turns the story into a boolean causal model, re-reading when the result is unusable, and an exact executor answers from that model. With a 14B model



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.

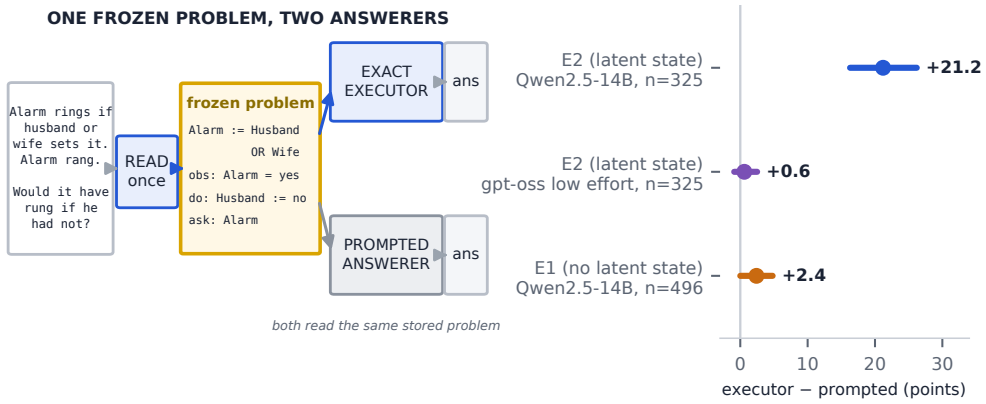


Figure 1: Left: the design, on the alarm story. The story is read once into a frozen problem, the extracted causal model stored and never changed afterwards, and that same stored object goes to both answerers. Right: the executor's accuracy advantage over the prompted answerer, in points, with bootstrap 95% intervals, on two of our problem sets (§4). In E2 the answer requires inferring latent state (root causes the story does not fix); there the advantage is large against Qwen2.5-14B and near zero against gpt-oss-120b at its lowest reasoning effort, on all problems. In E1 there is no latent state, and the advantage is not statistically resolved.

it matches the best published CounterBench result, obtained with a 671B model. We then use the separation to ask when exact computation helps.

To isolate the computation, we read each story once, freeze the resulting model, and give the identical model to two answerers (Figure 1): an exact executor, a program that checks every assignment of the model's root causes, and a prompted answerer, a language model that receives the same model as text together with the strongest prompt we built. Both read the model from the same stored copy, so any difference in accuracy is a difference in computing.

The alarm story shows the distinction that turns out to matter. Its answer depends on whether the wife set the alarm, which the story does not say. Her action is latent state: a root cause that the question does not fix and whose possible values must be inferred from the observations before the change is applied. Here the observation (the alarm rang) does not settle it, so the correct answer is "unsure"; had the story added "The wife did not set it", the answer would be "no". When there is no latent state, the answer follows by evaluating the rules forward, one after another, from known values.

We expected long causal chains to be where exact computation helps. They are not: on 496 stories with chains of up to 11 rules and no latent state, the executor shows no statistically resolved advantage. Latent state is where the answerers separate. On 325 problems that require inferring it, the executor is 21.2 points more accurate. The gap holds for a second model family, for a model more than twice the size, and for majority voting over five samples, and most of it disappears when the latent values are disclosed with everything else held fixed. On controlled problems the gap appears at every number of free roots and every depth we tested, and we detect no trend in either. A reasoning model, gpt-oss-120b, matches the executor's accuracy on the problems it answers at every reasoning effort we tried, and at its lowest effort on all of them: 95.4% of 325 against the executor's 96.0%. To answer an extracted problem it needs a median of 64 seconds and 1,376 generated tokens, against well under a millisecond and no generated tokens for the executor. Against a reasoning model, then, the executor's advantage at the answering stage is cost, not accuracy. All of this presupposes a correct reading: when the stories are reworded away from the benchmark's phrasing, the extraction loses information, and computing it exactly reproduces the loss.

Handing formalized problems to an external solver is established for logic and arithmetic (Pan et al., 2023; Ye et al., 2023; Olausson et al., 2023; Gao et al., 2023; Chen et al., 2023), and the three-step counterfactual recipe the executor implements is textbook causal inference (Pearl, 2009). What is new is a reader that makes exact counterfactual execution work from long causal stories, and an analysis of when it pays:

- A separated pipeline. A schema-constrained reader that translates long stories a few sentences at a time and re-reads when the executor cannot use its output, feeding an exact executor. With a 14B model it matches the best published CounterBench result, CoIn with the 671B DeepSeek-V3, and beats it on the split that requires inferring a latent cause; it also beats the strongest prompt we built at the same size.
- When exact execution helps. Freezing one extraction per item in a store that every answerer reads isolates the answering step. The result depends on the regime: little benefit without latent state; about 20 points with it against every non-reasoning answerer we tested; against a reasoning model, the same accuracy at a small fraction of the answering cost.
- The gap tracks the presence of latent state. It appears only when root causes are left free, at every number of free roots (one to ten) and depth (three to nine rules) we tested, with no detectable trend in either; and it shrinks when the latent values are disclosed while the graph, change, question and answer stay the same.
- A benchmark audit. Because the executor checks every world a story allows, it finds 60 official CounterBench answers that are inconsistent with their own stories.

## 2 Related Work

Translate, then solve. Logic-LM (Pan et al., 2023), SatLM (Ye et al., 2023) and LINC (Olausson et al., 2023) translate text into logic programs, constraints or first-order logic for an external solver; PAL (Gao et al., 2023) and Program of Thoughts (Chen et al., 2023) emit Python for an interpreter. Counterfactuals need two things these targets do not provide: an intervention, which cuts a variable off from its causes rather than constraining it, and an abduction step that infers the latent state before the intervention. The closest causal system, CausalTool (Hua et al., 2024), extracts a graph and routes the question to formula tools; its single-instance counterfactual tool propagates without inferring the unobserved roots, and the model writes the final answer. CounterBench's "Solver" baseline reimplements it from the paper's description and scores near chance for all ten models tested.

Where counterfactual reasoning fails. Gendron et al. (2024) find on CLadder that extracting the graph is largely solved while counterfactual prediction is the bottleneck, and Yang et al. (2026) evaluate the stages with correct inputs; in both, the language model performs every stage. Causal Cartographer (Gendron et al., 2025) separates extracting causal relationships from reasoning over them, but a constrained language-model agent still performs the counterfactual computation step by step. Executable Counterfactuals (Vashishtha et al., 2026) compares matched problems with and without a latent variable, reports a 25–40 point drop for o4-mini and Claude-4-Sonnet when one must be inferred, and improves models with supervised and reinforcement fine-tuning. Our disclosure experiment follows the same logic and adds one control: the problem is extracted once and frozen, so the two versions differ only in the added observation, and we measure how much of the executor's advantage the disclosure removes. Executable Counterfactuals asks whether training helps a model infer latent state. We ask whether, once the problem is fixed, the model needs to do the computation at all (Appendix Table 4). That question shows what a study of models alone cannot: with the problem fixed at the reader's own, imperfect extraction rather than a gold program, a non-reasoning answerer's loss is a failure of computation that a program removes, and removing it buys accuracy against non-reasoning answerers but only answering cost against a reasoning one.

## 3 Problem setting and method

Causal stories. A story describes boolean variables and gives a rule for each variable that has causes, built from AND, OR, a single cause and negation, for example Alarm = Husband ∨ Wife. Variables without causes are roots. The story may state observations, facts about what actually happened (Alarm = true). The question names an intervention, a hypothetical change that sets one variable to a value and cuts it off from its causes (do(Husband = false)), and a query, the variable whose value under that change is asked for. A world is an assignment of true or false to every root. Because the rules are deterministic, a world fixes the value of every other variable.

The correct answer. The standard recipe has three steps (Pearl, 2009; Balke and Pearl, 1994). Abduction keeps the worlds consistent with the observations. Action applies the intervention in each kept world. Prediction reads the query in each. If all kept worlds give the query the same value, the question is determined and that value is the answer. If they disagree, it is undetermined and the correct answer is "unsure". In the alarm story three worlds are consistent with the alarm ringing; once Husband is set to false, the alarm rings in two of them and stays silent in the third, so the answer is "unsure". The free roots are the roots the intervention does not set (here, Wife), and we write $R$ for their number. Their values are the latent state; observations may pin some of them down. $R$ differs from the number of roots the executor enumerates (every root when the observations describe the actual world, as all of ours do) and from the uncertainty left after conditioning on the observations; Appendix C.6 reports all three. When $R = 0$ there is nothing to infer, and the answer follows by evaluating the rules forward from the intervention; we call this forward propagation. The depth of a question is the number of rules on the longest chain from a root to the query.

Reading. The reader is a language-model call constrained by a JSON schema (Willard and Louf, 2023) to output the causal model: each rule (effect, causes, AND, OR or single cause, negations), the observations, the intervention, and the query with its polarity. It is instructed to extract the model without answering the question (Appendix A.3). A single call degrades on stories with more than about ten rules, so our chunked reader first copies out the causal sentences, translates them three at a time and merges the pieces; a final call reads the observations and the question. When the executor cannot use a reading, the system reads the story again with a single call (the fallback). Reader, fallback and executor together form the pipeline.

Exact execution. The executor carries out the three steps literally: it enumerates every world, keeps those consistent with the observations, applies the intervention and reads the query. Observations usually describe what actually happened and are checked in the actual world; CounterBench's conditional questions instead state a condition about the world after the change, and the executor checks those there (Appendix D.3). The executor returns one of four states: determined (with a value), underdetermined, inconsistent when no world is consistent with the observations, and invalid when the extracted model is malformed.

Proposition 1. Let $F$ be a finite, deterministic, acyclic set of boolean rules over variables $V$ with roots $U$, $O$ a set of observed values, do($X$=$x$) an intervention and $Q \in V$ a query. Let $W$ be the set of worlds (assignments to $U$) under which $O$ holds in the world where the question places it. The executor returns inconsistent if $W$ is empty; otherwise it returns determined with value $q$ if $Q = q$ in the intervened world for every member of $W$, and underdetermined if two members of $W$ give $Q$ different values. If $F$, $O$, $X$, $x$ and $Q$ are the ones the story states, these are the story's answers.

Proof. Because $F$ is acyclic and deterministic, each world fixes every variable, both without and with the intervention. The executor enumerates all $2^{|U|}$ worlds and keeps exactly those in $W$; the three cases are exhaustive and disjoint. A cyclic model, one with two rules for the same effect, or one missing the query or the intervention lies outside the assumptions and is returned as invalid. □

Table 1: Problem sets. $R$ is the number of free roots, whose values the answerer must infer.

| set | $n$ | what varies | used in |
|---|---|---|---|
| CounterBench alpha | 1,000 | four question types; 87% of items have $R = 0$ | §5.2, §5.1 |
| CounterBench backdoor | 200 | every item has an unobserved common cause ($R \geq 1$) | §5.1 |
| CLadder | 1,476 | everyday wording, template-generated (deterministic counterfactual split) | §5.6 |
| E1 | 496 | depth up to 11, $R = 0$ (a matched sample of the 2,000 generated) | §5.2 |
| E2 | 325 | $R \in \{1, 2, 3\}$, depth 3–9, how much the observations reveal; 96 undetermined | §5.3 |
| E2 reworded | 325 | the same problems in a different grammar | §5.6 |
| E2 disclosed | 148 pairs | the latent values stated as observations, or not | §5.3 |
| E3 | 280 | $R$ from 1 to 10 at fixed width; half undetermined | §5.4 |
| E4 | 144 | depth $\{3, 6, 9\} \times R \{1, 3, 6\}$; half undetermined | §5.4 |

Within this model class, then, every error the pipeline makes is a reading error. We check the implementation against two independently written references on 5,000 random models (Appendix D.1).

The matched comparison. To compare answerers rather than readers, the reader runs once per item and its output, the frozen problem, goes unchanged to every answerer. The prompted answerer receives it rendered as text, together with our strongest prompt: worked examples of the three-step recipe, tuned on a separate development split (Appendix A.3). Every answerer reads the problem from this shared store; the local runs record, for every item, that they read it there, and the executor's answers are computed from the same store. Scoring is strict and the same for every answerer: "yes" or "no" counts only when it matches a determined gold answer, and "unsure" only when the gold answer is "unsure". For the executor this means that an inconsistent or invalid state is always wrong. A prompted answer that is malformed, or that does not arrive within the 240-second limit per call, counts as no answer; counting the timed-out ones as correct instead changes no conclusion (Appendix B.3). The pipeline may read a story twice, so it is not part of this comparison; we report it separately as a system result. Two baselines skip the separate reading step: a one-pass prompt gives the model the raw story with worked examples of the same recipe and asks it to read and answer in one call, and chain-of-thought gives only an instruction to reason step by step (Wei et al., 2022) (Table 10 also gives it a perfectly parsed problem instead of the story).

## 4 Experimental setup

Problem sets. Table 1 lists them. CounterBench (Chen et al., 2026) and CLadder (Jin et al., 2023) are public benchmarks. We generated E1–E4 to vary one factor at a time. Every label in them is computed twice, by the executor and by an independently written brute-force reference, and an item is kept only if both agree. E1 and E2 were pre-registered: their construction and primary test were committed before the items existed. The frozen, strictly scored comparisons we report correct an input mismatch in those original tests (Appendix A.2). E3 and E4 are exploratory; the E4 analysis plan was committed before its items were generated.

Models. The reader and the local answerers are Qwen2.5-14B (Qwen Team, 2024) and Gemma-3-27B (Gemma Team, 2025), served by Ollama at temperature 0. We also run two larger answerers on the frozen E2 problems through a remote inference service: Gemma-4-31B (Gemma Team, 2026), with its thinking mode off, and gpt-oss-120b (OpenAI, 2025), a model trained for long reasoning, at its highest reasoning effort. Token budgets were fixed before each run (Appendix B.2).

Table 2: CounterBench official splits and official answer key, accuracy (%). All rows but the last use one model (Qwen2.5-14B) and our harness; CoIn is the benchmark's own prompting method. †As published.

| method | alpha | backdoor |
|---|---|---|
| chain-of-thought | 64.9 | 51.0 |
| CoIn (Chen et al., 2026) | 88.8 | 69.0 |
| strongest one-pass prompt | 89.4 | 79.0 |
| pipeline | 92.4 | 87.5 |
| template parser + executor | 94.5 | 87.5 |
| CoIn, DeepSeek-V3 (671B)† | 93.5 | 83.0 |

Statistics. Two answerers are compared on the same items. The discordant items are those exactly one of them gets right, and the exact McNemar test (McNemar, 1947) asks whether they split evenly. Intervals are bootstrap (Efron, 1979) 95% intervals on the accuracy difference, resampling items 10,000 times. Tests beyond the pre-registered ones are exploratory and not adjusted for multiplicity.

# 5 Results

## 5.1 The pipeline on CounterBench, and an audit of its answer key

With the same model (Qwen2.5-14B) and harness, the pipeline scores 92.4% on alpha against 89.4% for the strongest one-pass prompt we built (63 discordant items to 33, $p$=0.003; 91.9% against 89.6% on the 800 items outside the development sample, $p$=0.05), and 87.5% against 79.0% on backdoor (Table 2). The larger margin is on backdoor, where every item requires inferring an unobserved cause. Replacing our reader by a rule-based parser of CounterBench's fixed sentence templates, which reads perfectly, gives the executor's ceiling (last pipeline row). The 14B pipeline also matches the best published result, CoIn with DeepSeek-V3 (DeepSeek-AI, 2024), a 671B mixture-of-experts model with 37B active parameters: 91.6% against 91.8% over both splits, 92.4% against 93.5% on alpha and 87.5% against 83.0% on backdoor, as reported by Chen et al. (2026) in their own harness. The pipeline uses about four model calls per alpha item against one, twice the wall-clock time and 5–13× the prompt tokens (Appendix C.5); given more budget than that, the one-pass prompt still does not significantly exceed the executor (Appendix B.13).

Because the executor checks every world a story allows, it can also show when no world yields the official answer. Of 1,200 official items, 60 are contradictions, where the story forces one answer and the key gives the other, and 20 more cannot be evaluated as written. Each flag comes with a certificate that can be checked by hand, and all 80 recur when the models are rebuilt from the benchmark's separately released model files with a separately written parser. Under the two alternative semantics a reader might choose for conditional and joint questions, at least 75 of the 80 remain flagged (Appendices D.2, D.3). All scores above use the official key unchanged, so CounterBench differences of a few points carry this 5% key noise.

## 5.2 Without latent state, exact execution adds little

E1 asks whether exact execution helps as causal chains get longer when there is nothing to infer. On 496 items the prompted answerer scores 93.1% and the executor 95.6%, a difference of +2.4 points with 95% interval $[0.0, +4.8]$ (26 discordant items to 14, $p$=0.08). We find no statistically resolved advantage; this does not show that the two are equivalent. The pre-registered version of this test used an untuned prompt for the prompted answerer, which favoured the executor; it is reported in Appendix B.8.

CounterBench's alpha split, where 87% of items have no free root, agrees (Appendix Table 10). With the benchmark's fixed sentence templates parsed by a rule-based template parser, which recovers the exact model and so acts as a perfect reader, the prompted an-

Table 3: E2 accuracy (%) ($n$=325; 229 determined, 96 undetermined); every answerer may say "unsure". Only the prompted answerer and the executor form the matched comparison: they answer the same stored problem. The one-pass prompt reads the story itself, and the pipeline may read it twice.

| answering | all | determined | undetermined |
|---|---|---|---|
| one-pass prompt (reads and answers at once) | 62.5 | 73.8 | 35.4 |
| prompted answerer | 74.8 | 79.9 | 62.5 |
| exact executor | 96.0 | 95.6 | 96.9 |
| pipeline | 95.7 | 97.8 | 90.6 |

swerer and the executor score 93.1% and 94.5%. On 300 items read by our reader and frozen, they score 94.3% and 93.3%, not significantly different. Given the same perfect reading, a plain chain-of-thought instruction without our worked examples scores 62.9%, so the prompt matters. On such problems the gain comes from reading the story separately and prompting well, not from computing exactly. Across six readers, final accuracy tracks reading accuracy almost monotonically (Appendix, Figure 3).

### 5.3 With latent state, a large advantage

In E2 the observations constrain roots that the intervention leaves free, so the answer depends on first working out which of their values are consistent with what was observed. Here the answerers separate (Table 3). From the same frozen problems, the executor scores 96.0% and the prompted answerer 74.8%, a difference of +21.2 points with 95% interval $[+16.3, +26.2]$ (74 discordant items to 5). The executor is ahead both on determined problems (95.6% against 79.9%) and on undetermined ones, where it reports 93 of 96 as open and the prompted answerer says "unsure" on 60. In E2 the gap is also larger with more free roots (+14.5, +20.5 and +30.8 points at $R = 1$, 2 and 3), although §5.4 shows that this rise does not continue on controlled problems.

Two controls locate the gap. First, it is in the computing, not the reading: given the generator's own correct problem instead of our extraction, the prompted answerer rises only to 76.9% (12 discordant items to 5, $p$=0.14), and stays 23.1 points (95% interval $[18.5, 27.7]$) below the executor on the same gold problems, where the executor is exact by construction. Second, it depends on the latent state. We took the 148 determined problems that leave at least two assignments of the free roots consistent with the observations, and disclosed one consistent assignment as an extra observation; the graph, intervention, query and answer are unchanged. The prompted answerer rose from 83.8% to 95.3% while the executor stayed at 100%, so the gap fell from +16.2 to +4.7 points (change −11.5, 95% interval $[-18.2, -4.7]$). Disclosure also adds information, so this shows that the gap depends on the latent state; it does not identify how the prompted answerer computes (Appendix B.4). Finally, the gap is not an artifact of our prompt having been tuned on E1: a prompt written for this inference and developed on a separate split did worse (51.1%), mostly by answering "unsure" on problems the story determines (Appendix A.4).

### 5.4 More latent state, and depth crossed with it

E2 leaves at most three roots free. E3 raises the number of free roots to ten while holding the graph at 14 rules and the query's depth between 4 and 8; at every $R$ half the problems are undetermined, so answering "unsure" every time scores 50%. E4 crosses depth $\{3, 6, 9\}$ with $R \in \{1, 3, 6\}$, 16 problems per cell. In both, every answerer receives the generator's own problem, so no reading error enters (Figure 2).

The executor is correct on every E3 and E4 problem by construction, because each label is the executor's own answer, confirmed by an independent reference; the informative quantities are the other answerers' accuracy and cost. The advantage over the prompted answerer is present throughout, and we detect no growth with the amount of latent state or with depth. In E3 the prompted answerer scores between 57.5% and 75.0% at every $R$, with no trend (permutation test, $p$=0.84), and the gap to the executor is +33.2 points overall. In

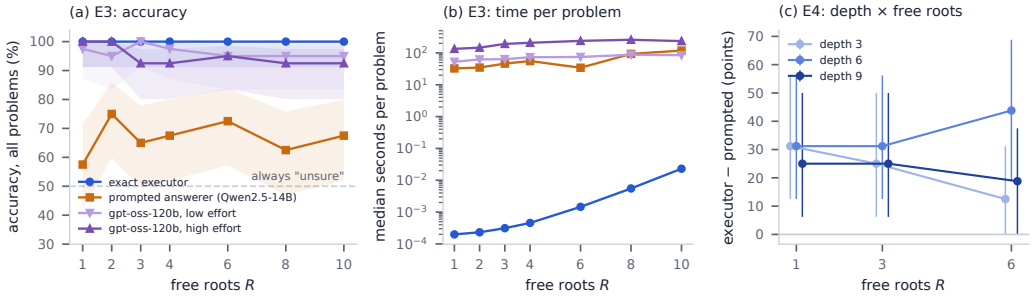


Figure 2: E3 and E4, with every answerer given the generator's own problem. (a) Accuracy on all problems against the number of free roots $R$, with Wilson (Wilson, 1927) 95% intervals; half the problems at each $R$ are undetermined, so answering "unsure" every time scores 50%. (b) Median wall-clock seconds per problem, log scale; indicative only, because the local model shared a GPU and gpt-oss-120b ran on a shared remote service. (c) E4: the executor's margin over the prompted answerer (Qwen2.5-14B), with bootstrap 95% intervals, one line per depth.

E4 the gap is +27.1 points overall and positive in all nine cells (smallest +12.5). By $R$ it is +29.2, +27.1 and +25.0; by depth +22.9, +35.4 and +22.9. Neither trend is statistically resolved (stratified permutation tests fixed before the items were generated: $p$=0.682 for $R$ within depth, $p$=1.00 for depth within $R$). With 16 problems per cell E4 rules out only large trends: the margin at $R$=6 minus that at $R$=1 is −4.2 points (95% interval $[-20.8, +14.6]$), and depth 9 minus depth 3 lies in $[-16.7, +16.7]$. The rise from $R$=1 to $R$=3 in E2 does not carry over to these controlled sets; what we can say is that the answerers separate whenever there is latent state to infer, at every amount and depth we tested (Appendix B.7 bounds the effect of replies that ran past the token cap).

The reasoning model also holds up. At low effort gpt-oss-120b answers every E3 problem and scores at least 95.0% at every $R$, while its median generated tokens rise from 1,366 at $R$=1 to 1,934 at $R$=10; at high effort they rise from 3,199 to 6,079, and the budget cuts off 14 of the 280 problems. The executor's time grows exponentially with $R$, as enumeration must, from 0.20 ms to 23.0 ms per problem, still far below a single model call (Appendix B.7).

### 5.5 Stronger answerers

We replace the prompted answerer while keeping the frozen E2 problems and the prompt fixed (Table 5). Gemma-4-31B, more than twice the size of our reader, trails the executor by +20.3 points on the 306 problems it completes; it ran with thinking mode off, through raw completions in its own turn format, because the deployment exposed no chat template. Gemma-3-27B, reading the stories itself and answering from its own frozen extraction, trails by +24.3 (81 discordant items to 2). Sampling Gemma-4-31B five times at temperature 0.7 and taking the majority (self-consistency; Wang et al., 2023) costs 4.0 times the generated tokens of one pass; it moves accuracy from 71.1% to 75.4% ($p$=0.15) and leaves it +20.6 points behind the executor (72 discordant items to 5).

gpt-oss-120b, a model trained for long reasoning, matches the executor on the problems it answers at every reasoning effort we tried (Table 5), with a 16,384-token budget fixed before the runs; on all problems, higher effort loses points to that budget. At its lowest effort it answers 321 of the 325 problems and scores 95.4% on all of them, against the executor's 96.0% (difference +0.6 points, 95% interval $[-0.9, +2.5]$). Scored against what the frozen problem implies rather than against the story, gpt-oss-120b at high effort agrees with the executor on 99.3% of its answered problems and Qwen2.5-14B on 75.5%: the non-reasoning answerers fail at computing the problem they are given, and the reasoning model does not (Appendix B.2). What separates the reasoning model from the executor is cost. At low effort it needs a median of 64 seconds and 1,376 generated tokens per problem; at high effort 161 seconds and 3,839 tokens, and the budget cuts off 17 problems. The executor generates

no tokens and takes a median of 0.11 ms on one CPU core. Our first answer parser accepted only the requested answer format and scored many low-effort replies as unreadable; every gpt-oss-120b number here uses the corrected parser (Appendix B.2).

### 5.6 When the reading fails

Exact computation is only as good as the problem it is given. We rewrote every E2 problem from its own gold model into a different grammar, with renamed variables, restated rules, shuffled sentence order and reworded questions; the content is unchanged by construction, and the benchmark's template parser cannot read any of it (Appendix B.5). Our chunked reader degrades: it recovers the exact set of rules on 45% of items against 82.8% before, and on 163 of 325 problems the frozen extraction no longer determines an answer. The pipeline reads those stories again and keeps a large margin (91.4% against 70.8%). On the frozen extraction, however, the margin falls to +6.8 points (63 discordant items to 41), and on the stories that do determine an answer the executor is no better than the prompted answerer (69.0% against 72.9%). A reading that drops a rule or an observation leaves roots free that the story pins down; the executor then correctly reports the extracted problem as open, which is wrong for the story (67 of the 229 determined stories). The prompted answerer does not follow the extracted problem as closely: its answer differs from the one the problem implies on 119 items, and on 41 of those it is nonetheless right about the story. Exact computation helps only as far as the extracted problem is right.

CLadder, whose template-generated stories use everyday wording, shows the same limit. The pipeline scores 86.7% and a one-pass prompt 89.0% ($p$=0.04; not significant on the 1,326 items outside the development sample). The chunked reader alone scores 15.9%, because its sentence splitter does not fit CLadder's style, and the fallback handles 1,241 of 1,476 items. On these problems the bottleneck is reading, the complementary failure to E2's.

## 6 Limitations

The advantage depends on the answerer. It is large against every non-reasoning answerer we tried; against a reasoning model it is a matter of cost alone, which cheaper reasoning could narrow. The reader limits transfer. Exact computation cannot restore information the reader lost, as the reworded stories and CLadder show. The formalism is narrow. We handle explicit deterministic rules built from AND, OR, single causes and negation. Thresholds, counts and hedged rules fall outside it, and enumeration grows exponentially with the number of free roots, so much larger latent spaces would call for posing abduction as a satisfiability problem. Our problem sets are synthetic. E1–E4 are generated by us, and all runs are single runs. Descriptive comparisons. Reasoning and non-reasoning answerers differ in family, size and serving, so their comparison describes these models and does not isolate the effect of reasoning training. Wall-clock figures come from our hardware and a shared service and are indicative only. Models and prompts. We test open-weight models only, and the prompted answerer uses prompts we wrote; better prompts or proprietary models could narrow the gap.

## 7 Conclusion

A counterfactual question about an explicit story poses two problems: reading the story and computing the answer. Separating them lets a 14B model match a 671B model on CounterBench. With the reading fixed, a well-prompted model propagates forward from known values about as well as an exact executor, which is then simply the cheaper choice. When the answer requires inferring latent state, the executor is 20 to 30 points more accurate than every non-reasoning model we tried and as accurate as a reasoning model, without generating a token, and each of its answers comes with the set of worlds that justifies it. These are costs of answering an extracted problem; reading the story is a separate cost, and when the reading fails, exact computation cannot help.

AI use statement

The research idea, the questions this paper asks and every claim it makes are the authors'. Generative AI tools, used under the authors' direction, assisted with implementing the code, running the experiments, analysing the results, and drafting and revising the text. The authors verified every reported number against source code and raw results, and take full responsibility for the paper.

Ethics statement

This work uses only public benchmarks and open-weight models. Most were run locally; Gemma-4-31B and gpt-oss-120b, both open-weight, were queried on a remote inference service. No human subjects or personal data are involved. The audit flags benchmark answer keys that the executor cannot reconcile with their stories; we release it for community adjudication and adjust no score by it, so that the benchmark authors and other users can verify each flag independently; we will send the audit to the benchmark authors once the review period ends.

Reproducibility statement

Every number in this paper traces to a committed per-item JSON artifact (prediction, gold, extraction flags for every item of every reported run, more than 5,000 items). We distinguish two kinds of reproduction. Recomputing the metrics from those artifacts is deterministic: one registry script computes every E2 number, the paper prints them through generated macros, and a verification script re-derives them independently from the raw rows. Re-running inference is expected to reproduce the results closely but not bit for bit: local runs use temperature 0 through Ollama, whose outputs can vary across hardware and versions, the five-sample vote uses temperature 0.7 with fixed seeds, and remote outputs depend on the serving stack. The supplementary material, a single archive (supplementary_material.zip) with a README.md at its root, contains the executor, both readers, all baseline scripts, the generators and independent label checkers for the CB-Hard and E1–E4 sets, our generated problem sets, the per-item outputs of every run, the pre-registrations and the E4 analysis plan, the property tests that check the executor against independent references, the audit script that re-derives every flagged item from its story text, a script that re-derives every number in this paper from the artifacts, and exact reproduction commands. Every model is open-weight. The reader and the local answerers run through Ollama; Gemma-4-31B and gpt-oss-120b were queried on a remote inference service, whose address and key the released code reads from the environment. The released code records the model identifier, reasoning effort, token budget, finish reason and token counts of every remote call, so a rerun on another deployment of the same weights can be compared item by item. Its README gives the commands that recompute every number in the paper from the included outputs, without model calls, in about a minute; every appendix that quotes a script names the file it comes from. We will release the full repository publicly upon acceptance, including the commit history that timestamps the pre-registrations.

## References

A. Balke, J. Pearl. 1994. Probabilistic Evaluation of Counterfactual Queries. In AAAI.

W. Chen, X. Ma, X. Wang, W. W. Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. TMLR.

Y. Chen, V. K. Singh, J. Ma, R. Tang. 2026. CounterBench: Evaluating and Improving Counterfactual Reasoning in Large Language Models. In AAAI. (arXiv:2502.11008)

DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. arXiv:2412.19437.

B. Efron. 1979. Bootstrap Methods: Another Look at the Jackknife. The Annals of Statistics 7(1):1–26.

R. El-Yaniv, Y. Wiener. 2010. On the Foundations of Noise-free Selective Classification. JMLR 11:1605–1641.

L. Gao et al. 2023. PAL: Program-Aided Language Models. In ICML.

Gemma Team. 2025. Gemma 3 Technical Report. arXiv:2503.19786.

Gemma Team. 2026. Gemma 4 Technical Report. arXiv:2607.02770.

G. Gendron, J. Rožanec, M. Witbrock, G. Dobbie. 2024. Counterfactual Causal Inference in Natural Language with Large Language Models. arXiv:2410.06392.

G. Gendron, J. M. Rožanec, M. Witbrock, G. Dobbie. 2025. Causal Cartographer: From Mapping to Reasoning Over Counterfactual Worlds. arXiv:2505.14396.

Z. Hua, S. Xing, H. Jiang, C. Wei, X. Wang. 2024. Improving Causal Inference of Large Language Models with SCM Tools. In NLPCC.

Z. Jin et al. 2023. CLadder: Assessing Causal Reasoning in Language Models. In NeurIPS.

Q. McNemar. 1947. Note on the Sampling Error of the Difference between Correlated Proportions or Percentages. Psychometrika 12(2):153–157.

T. Olausson et al. 2023. LINC: A Neurosymbolic Approach for Logical Reasoning by Combining Language Models with First-Order Logic Provers. In EMNLP.

OpenAI. 2025. gpt-oss-120b & gpt-oss-20b Model Card. arXiv:2508.10925.

L. Pan, A. Albalak, X. Wang, W. Wang. 2023. Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning. In Findings of EMNLP.

J. Pearl. 2009. Causality: Models, Reasoning, and Inference, 2nd ed. Cambridge University Press.

Qwen Team. 2024. Qwen2.5 Technical Report. arXiv:2412.15115.

A. Vashishtha et al. 2026. Executable Counterfactuals: Improving LLMs' Causal Reasoning Through Code. In ICLR. (arXiv:2510.01539)

X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In ICLR.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, D. Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In NeurIPS.

B. T. Willard, R. Louf. 2023. Efficient Guided Generation for Large Language Models. arXiv:2307.09702.

E. B. Wilson. 1927. Probable Inference, the Law of Succession, and Statistical Inference. Journal of the American Statistical Association 22(158):209–212.

K. Yamin, S. Gupta, D. Ghosal, Z. Lipton, B. Wilder. 2024. Failure Modes of LLMs for Causal Reasoning on Narratives. arXiv:2410.23884.

S. Yang, Q. Yang, L. Tang, Y. Meng, N. Guo, J. Blackburn, Z. Xi. 2026. On the Eligibility of LLMs for Counterfactual Reasoning: A Decompositional Study. In ICLR.

X. Ye, Q. Chen, I. Dillig, G. Durrett. 2023. SatLM: Satisfiability-Aided Language Models Using Declarative Prompting. In NeurIPS.

Guide to the appendix. Appendix A gives the protocols: which items each comparison uses, the prompts, and how the code was produced. Appendix B gives every matched comparison in full, including the stronger answerers, the disclosure experiment, the second model family, the reworded stories and the per-stratum E2 results. Appendix C covers reading, abstention and cost. Appendix D validates the executor and documents the benchmark audit. Two further generated sets appear only here: CB-Hard (200 stories in CounterBench's style with 8–14 variables) and CB-Hard-XL (200 with 15–30 variables), which stress the reader on long stories. Terms are those of §3: the prompted answerer receives the frozen problem; the one-pass prompt reads the raw story itself; chain-of-thought is a plain step-by-step instruction, given the story unless a table says otherwise; the pipeline is reader plus executor with the fallback re-read.

# A Protocols, prompts and provenance

## A.1 Nearest prior work at a glance

Table 4: The question each line of nearest prior work asks, whether it runs an exact executor, and whether it gives the same extracted problem to the answerers it compares.

| work | question studied | executor | same problem |
|---|---|---|---|
| Logic-LM, SatLM, LINC | does a solver improve deductive QA over prompting? | yes | no |
| PAL, Program of Thoughts | does an interpreter improve arithmetic QA? | yes | no |
| CausalTool | can formula tools answer causal queries from text? | partial | no |
| Gendron et al. (2024); Yang et al. (2026) | which stage of counterfactual reasoning fails? | no | partial |
| Causal Cartographer | can an agent reason over extracted causal relationships? | no | partial |
| Executable Counterfactuals | does latent state make counterfactuals harder, and does training help? | no | no |
| this paper | when does exact execution of a fixed problem beat a prompted answerer? | yes | yes |

## A.2 Which comparison uses which items

Every comparison in the paper, with its sample and its status. "Frozen" means run once after prompt development was closed; "pre-registered" means the analysis plan was committed before the items existed; "correction" means added after the pre-registered run, to fix an asymmetry we found in it; "confirmatory" means run after the analysis it repeats was already fixed. The matched E1 row is the 496-item extension; it supersedes the 120-item version, which is retained in the artifacts. How the matched rows are built. Every matched comparison reuses one cached extraction per item. The cache holds the chunked extraction. Where it was unusable the pipeline re-read with the single-pass fallback and answered from that reading while the other arms kept the chunked one, so on those items (104 of 325 on E2) the pipeline's own figure is not an answering-only comparison. Every matched claim in §5.3 is therefore computed with the executor held to the same cached reading the other arms received, on all 325 items, and the pipeline's number is reported separately as the system result (Appendix B.1). Items are taken in the generator's order up to the budget and never selected on any arm's outcome. The E1 depth comparison originally could not be verified this way—its prompted answerer re-read rather than loading a cache and neither arm persisted its extraction—so we re-ran it with experiments/run_e1_matched.py, which reads once per item, resolves the retry before either arm answers (22 items), and hands both the same object. The numbers in §5.2 are from that re-run.

| set | $n$ | what it compares | status |
|---|---|---|---|
| alpha (official) | 1,000 | ours vs. prompting, CoIn, oracle | frozen |
| backdoor (official) | 200 | same, with a confounder | frozen |
| alpha, matched | 300 | one cached extraction, executor vs. in-text | correction |
| depth sets (E1) | 2,000 | ours vs. tuned and untuned prompts | pre-registered |
| E1, matched | 496 | one frozen problem per item, retry resolved first | repaired |
| E2, second family | 325 | Gemma-3-27B reads and answers | confirmatory |
| abduction sets (E2) | 325 | depth × free roots × evidence | pre-registered |
| E2, frozen input | 325 | one cached extraction, strict executor scoring | correction |
| E2, other answerers | 325 | Gemma-4-31B, 5-sample vote, gpt-oss-120b | exploratory |
| latent disclosure | 148 | same problem with the latent roots stated | exploratory |
| E3 | 280 | free roots $R$ up to 10 | exploratory |
| CLadder | 1,476 | transfer to everyday wording | frozen |
| reader ablation | 2,476 | single-pass vs. chunked vs. +fallback | frozen |
| paraphrases | 189 | reworded reading | exploratory |

### A.3 Extraction prompt (Stage 1)

The instruction text of the single-pass reader, quoted from the released code (the chunked reader uses the same rules on statement subsets, plus a copy-only splitting call and a question-only call). Nothing else is sent to the model; there is no hidden system prompt. What follows is an excerpt: the rules block is shortened for the page, and the complete prompt and JSON schema are the file experiments/run_neurosym.py in the released code.

```
You translate a natural-language structural causal model + a counterfactual
question into a JSON object. Do NOT reason about the answer; only extract the
structure exactly as written.
Each mechanism = one caused variable (the "target") and how its causes combine.
CRITICAL - where the word "not" goes:
- "A causes not B"  -> the "not" is on the EFFECT B.
- "not A causes B"  -> the "not" is on the CAUSE A.
Never set both input-negated and output_negated for a single "not".
Combining causes: "A and B together cause C" -> AND; "A or B causes C" -> OR;
a single cause -> IDENTITY. Only include causes explicitly joined to THAT target.
"A causes B, C and D" (one cause, several effects) -> one mechanism per effect.
OBSERVATIONS ("We observed ..."): "We observed X" -> {X, true};
"not X" -> {X, false}. Split multiple on "and".
CANONICAL VARIABLE NAMES: use ONE positive base name per variable everywhere;
internal negation ("alarm not set by wife") refers to the positive base
variable with a negation flag.
QUESTION -> interventions (the counterfactual antecedent), query_variable, and
query_polarity ("Would the room be bright?" where the base variable is
dark_room -> query_variable dark_room, polarity false).
WORKED EXAMPLES:
INPUT given="We know that Ziklo causes not Blaf, Blaf and Trune together cause
  Vork, Vork causes Lumbo. We observed Trune" question="Would Lumbo occur if
  not Ziklo instead of Ziklo?"
OUTPUT {"mechanisms":[{"target":"Blaf","operator":"IDENTITY","inputs":[{"variable":
  "Ziklo","negated":false}],"output_negated":true},{"target":"Vork","operator":"AND",
  "inputs":[{"variable":"Blaf","negated":false},{"variable":"Trune","negated":false}],
  "output_negated":false},{"target":"Lumbo","operator":"IDENTITY","inputs":[{"variable":
  "Vork","negated":false}],"output_negated":false}],"observations":[{"variable":"Trune",
  "value":true}],"interventions":[{"variable":"Ziklo","value":false}],
  "query_variable":"Lumbo","query_polarity":true}
INPUT given="We know that husband sets alarm causes alarm not set by wife. husband
  sets alarm or alarm set by wife causes ringing alarm." question="Would the alarm
  doesn't ring if not husband sets alarm instead of husband sets alarm?"
OUTPUT {"mechanisms":[{"target":"alarm set by wife","operator":"IDENTITY","inputs":
  [{"variable":"husband sets alarm","negated":false}],"output_negated":true},{"target":
  "ringing alarm","operator":"OR","inputs":[{"variable":"husband sets alarm","negated":
  false},{"variable":"alarm set by wife","negated":false}],"output_negated":false}],
  "observations":[],"interventions":[{"variable":"husband sets alarm","value":false}],
  "query_variable":"ringing alarm","query_polarity":false}
```

(Line breaks inside the two examples are added here for the page; the released file has each on one line. As noted above, the rules block is an excerpt.) Both worked examples are development-split items (§4; the registry is released), which is why every headline comparison is also reported on the untouched remainder. CLadder's stories are template-generated, so the second example necessarily shares its template with scored items; the one-pass prompt's worked examples are drawn from the same development pools. The one-pass prompt receives an explicit read–abduct–intervene–propagate algorithm plus two

executor-verified worked traces and answers under the same JSON schema; its full text is likewise in the released code.

### A.4 A prompt built for abduction

The prompted answerer of §5.3 uses the depth-tuned prompt, which was never developed for abduction, so its 21-point deficit could be a prompt artifact. We tested that. The new prompt states the procedure outright—list the free variables, enumerate their assignments, keep those consistent with the facts observed before the change, apply the change, read the query, and answer "unsure" only when the kept assignments disagree—with one determined and one undetermined worked trace and a note that a free variable which cannot reach the query cannot change the answer. It was developed on a separate 81-item pool (different generator seed; no story shared with the test set), revised at most twice and judged only on that pool, then frozen; the protocol, the development log and the hashes are released.

On the 325 test items, given the same frozen extraction, it scores 51.1% against the depth-tuned prompt's 74.8% and the executor's 96.0% (152 discordant items to 6 against the executor, $p<10^{-36}$). The margin holds at every number of free roots: +40.2 points at one, +47.9 at two, +47.3 at three. Handed the template parser's problem instead, so that no reading error reaches it, it scores 53.8%: reading error accounts for 2.7 of the 44.9 points, and the rest is the answering.

The reason it does worse is the interesting part. Teaching a prompt to recognise genuine ambiguity makes it decline far more often: it answers "unsure" on 180 of 325 items against the depth-tuned prompt's 88, including 113 of the 229 items whose story does determine an answer. It gains on genuinely open items (69.8% against 62.5%) and collapses on determined ones (43.2% against 79.9%). In our development pool a third version that forced the enumeration into the output reversed the trade, lifting determined accuracy to 59.3% while dropping undetermined accuracy from 95.5% to 45.5%. A prompted arm cannot reliably separate "the enumeration defeated me" from "the story leaves this open"; the executor gets that distinction for free (Appendix C.4). None of this shows that no prompt could close the gap—a finite search cannot—but it is the strongest attempt we could make, and it made the prompted arm worse.

### A.5 Implementation provenance

CoIn. The benchmark's own method; its code is unreleased, so we implemented it from the paper's description: build the causal graph in text, propose an answer, check it against the story, backtrack on a failed check, up to the paper's iteration cap. We keep the method name because the procedure is the one described, and we note the two places the description underdetermines it (the backtracking trigger and the stopping rule), which we resolved in the direction most favourable to the method. On the 200-item alpha sample at 14B it scores 87.5%.

Scaffolded prompt. Ours, not from the literature: it spells out the read–abduct–intervene–propagate recipe with executor-verified worked examples. We report it as the strongest prompting baseline we could build rather than as a reproduction of any published method.

Published rows. Taken from the benchmark papers as printed; we ran nothing for them.

Models. Qwen2.5-{7B,14B}, Gemma-2-{2B,9B}, Gemma-3-27B, Llama-3.1-8B and DeepSeek-R1-Distill-Qwen-14B, all 4-bit quantised, served locally at temperature 0.

## B The matched comparisons

### B.1 How the matched E2 comparison is built, and what it excludes

The problem. When the executor cannot commit to an answer from the chunked reading, the pipeline reads the story a second time and answers from that. This happened on 104 of the 325 E2 items. Only the first reading was stored, so comparing the pipeline with the

prompted answerer, which received that first reading, would change the input as well as the answerer on those 104 items.

A repair we rejected. Keeping only the 221 items where no second reading happened equalises the input, but it also selects items: the second reading is triggered exactly when the first leaves the query open, so the subset holds 220 determined items and one undetermined item. It says nothing about undetermined problems.

The repair we use. The executor answers from the stored first reading too, so every answerer consumes the same problem on all 325 items. The match holds by construction: the executor's answers are computed from the stored extraction, the local prompted runs record for each of the 325 items that they read their input from the store rather than re-reading the story, and the remote runner loads the store for every item. A per-item hash of the stored problem (rules with operators and negations, observations with values, which world the observations refer to, the intervention, the query and its polarity; experiments/provenance_audit.py) documents what that shared input was. Because it is computed from the store rather than from separately logged prompts, it records the shared input but does not independently prove what each run sent; logging a hash of each prompt as sent would. The same audit shows why the pipeline cannot be entered into a matched table at all: on the items where it re-read, the extraction it answered from was never persisted, so what it consumed is not recoverable from the artifacts.

The resulting accuracies are those of Table 3. The matched difference is +21.2 points, 74 discordant items to 5 ($p<10^{-16}$): +15.7 on the determined stories and +34.4 on the undetermined ones. It is a difference in answering under a fixed reader, not a reader-independent effect: a different reader could produce different problems. The frozen executor beats the pipeline on the undetermined stories (96.9% against 90.6%): reading a genuinely open story a second time sometimes returns a confident answer, a cost of the retry policy rather than of execution. Produced by experiments/e2_registry.py.

### B.2 Every answerer on the frozen E2 problems

Every row of Table 5 answers the same frozen extraction with the same depth-tuned prompt; only the model changes, and budgets were fixed before each run. Gemma-4-31B and gpt-oss-120b were served remotely. The Gemma deployment exposed no chat template, so we queried it through raw completions in its own turn format, with thinking mode off and a 4,096-token budget; the 19 problems it left unanswered hit that budget. gpt-oss-120b ran through its chat interface at temperature 0 with a 16,384-token budget at each of its three reasoning efforts. A problem whose answer cannot be read, or whose budget runs out, is recorded as unanswered; the table reports each answerer both on the problems it answered and on all of them.

Table 5: Answerers on the 325 frozen E2 problems (accuracy %, gap = executor minus answerer, in points).

| | | problems it answered | | | all problems | | |
|---|---|---|---|---|---|---|---|
| answerer | answered | acc. | executor | gap | acc. | gap | fidelity |
| Qwen2.5-14B | 325 | 74.8 | 96.0 | +21.2 | 74.8 | +21.2 | 75.5 |
| Gemma-4-31B | 306 | 75.5 | 95.8 | +20.3 | 71.1 | +24.9 | 75.0 |
| Gemma-4-31B, vote of 5 | 325 | 75.4 | 96.0 | +20.6 | 75.4 | +20.6 | 76.5 |
| gpt-oss-120b, low | 321 | 96.6 | 96.0 | −0.6 | 95.4 | +0.6 | 98.7 |
| gpt-oss-120b, medium | 314 | 96.8 | 95.9 | −1.0 | 93.5 | +2.5 | 99.0 |
| gpt-oss-120b, high | 308 | 96.4 | 96.1 | −0.3 | 91.4 | +4.6 | 99.3 |

Fidelity is agreement with the answer the frozen problem implies (yes or no where the executor finds it determined, unsure where it finds it open), as opposed to agreement with the story's gold answer; the executor attains it by construction. Problems whose extraction is malformed have no implied answer and are left out. The non-reasoning answerers reach about three quarters and gpt-oss-120b about 99% at every effort on the problems it answers, so the gap is a failure to compute the given problem. On the reworded stories (Appendix B.5)

the prompted answerer's fidelity drops to 63.2%, and 41 of its departures from the extracted problem happen to match the story. Median generated tokens per problem are 480 for Gemma-4-31B, 2,187 for the vote of 5, and 1,376, 1,946 and 3,839 for gpt-oss-120b at low, medium and high effort; the executor generates none and takes a median of 0.11 ms. At high effort gpt-oss-120b takes a median of 161 seconds per problem (90th percentile 277).

Reading the answer. The prompt asks for a final line ANSWER: yes|no|unsure. At low and medium effort gpt-oss-120b often writes a heading instead (**Answer**, with the word on the next line). Our first parser accepted only the requested format and scored such replies as unreadable: 152 of 325 at low effort and 131 at medium. We now read the answer with a fixed three-step rule (experiments/answer_parse.py): the requested format if present; otherwise the word "answer" followed only by punctuation or whitespace and then yes, no or unsure; otherwise a reply whose last line is the bare word. A reply the first parser could read keeps its answer. The replies the rule recovers are about as accurate as those read the first time (95.4% against 97.6% at low effort, 96.2% against 97.3% at medium), so it reads answers rather than guessing them. The low- and medium-effort runs stored every unreadable reply and were re-read without new model calls; the first high-effort run did not store its replies, so we repeated it with identical settings and report the repeat. Re-asked with twice the budget, all 30 problems the first high-effort run left unanswered were answered, 29 correctly; this diagnostic enters no reported number.

### B.3 Time limits and malformed outputs

Every local prompted call ran with a 240-second client-side limit and no token limit. At our generation speed that is roughly 3,800 generated tokens, far beyond a normal answer, so a call that hits it has usually entered a repeating loop. Such a call, like one whose output is not valid JSON, is scored as no answer. The executor has no counterpart failure, so this choice can only favour it. To bound the effect we recount every timed-out item as correct for the prompted answerer: E1 moves from +2.4 to +2.0 points, the gold-problem control from +23.1 to +21.5, Gemma-3-27B from +24.3 to +24.0, and the reworded stories from +6.8 to +4.6; the main E2 comparison had no timeouts (its 5 failures are malformed outputs). No conclusion changes. E3 and E4 ran while two jobs shared the machine, so load could make the time limit bind; we re-issued the three E3 calls that timed out and ran every later E3 and E4 call with the limit's token equivalent instead (3,840 generated tokens, no time limit).

### B.4 Disclosing the latent state

Eligible problems are the 229 E2 items whose story determines the answer, selected on the generator's semantics and never on any model's output. For each we enumerate the assignments of the roots the intervention leaves free that are consistent with the evidence, and disclose the first in lexicographic order of root names as additional evidence. Only the free roots are disclosed: together with the intervention their values fix the answer, so the disclosure removes all latent uncertainty, while the intervened root's actual value does not affect the answer. Disclosing it would also be invalid on problems without observations, whose evidence is checked in the world after the change, where a stated actual value of the intervened root would conflict with the intervention. Problems with fewer than two consistent assignments are dropped, since there is nothing to disclose, leaving 149; one was lost to a transport error, leaving 148. Every pair is checked with the executor on the problem as disclosed, and the gold answer holds in both versions on every pair. Both versions are built from the generator's own problem and supplied identically to both answerers as structured evidence.

The prompt answers 83.8% of the latent versions and 95.3% of the revealed ones; 22 problems flip to right and 5 to wrong ($p$=0.002). The executor is right on all of them in both versions, so the gap falls from +16.2 to +4.7 points, a paired change of −11.5 with 95% interval $[-18.2, -4.7]$ computed at the problem level. Produced by experiments/gen_revealed_state.py and experiments/run_revealed_state.py.

### B.5 A second model family, and reworded stories

Both experiments repeat E2 with one element changed (Table 6). In the first, Gemma-3-27B reads the 325 stories itself and answers from its own frozen reading; nothing is shared with the Qwen run. In the second, every story is re-rendered from its own gold model: variables renamed from a disjoint nonsense pool, rules written as "C fails to hold whenever both A and B do" instead of "A and B together cause not C", statements shuffled, observations as "It was in fact the case that X held" and questions as "Suppose, contrary to fact, that X had not held. Would Q have held?". Three checks pass on all 325 items: an independently written parser for the new grammar recovers the gold model exactly, the executor on that model reproduces the gold answer, and the benchmark's template parser cannot read any of them.

Table 6: E2 read by a second model, and E2 reworded (accuracy %; one frozen extraction per setting).

| | Gemma-3-27B reads and answers | | | reworded stories, Qwen2.5-14B | | |
|---|---|---|---|---|---|---|
| | all | det. | undet. | all | det. | undet. |
| one-pass prompt | 66.8 | 79.9 | 35.4 | 67.1 | 75.5 | 46.9 |
| prompted answerer | 74.5 | 83.4 | 53.1 | 70.8 | 72.9 | 65.6 |
| exact executor | 98.8 | 98.7 | 99.0 | 77.5 | 69.0 | 97.9 |
| pipeline (may read twice) | 96.9 | 99.1 | 91.7 | 91.4 | 91.7 | 90.6 |
| executor − prompted | +24.3 | +15.3 | +45.8 | +6.8 | −3.9 | +32.3 |
| discordant items | 81 v 2 | 36 v 1 | 45 v 1 | 63 v 41 | 32 v 41 | 31 v 0 |

Second family. The margin replicates: +24.3 points for Gemma against +21.2 for Qwen, on both the determined and the undetermined stories. On E2 it is larger at more free roots in both families (+14.5, +20.5, +30.8 on Qwen and +20.5, +23.1, +30.8 on Gemma for $R$=1, 2, 3); on the controlled E3 and E4 problems that rise does not continue (§5.4). The larger model's prompt is the stronger of the two on the determined stories (83.4% against 79.9%), and its margin is nonetheless the wider.

Reworded stories. The reader recovers the exact rule set on 44.9% of items against 82.8% on the original phrasing, measured against the renamed gold model, and the fallback fires on 163 items against 104. The pipeline loses little accuracy, because most rule errors do not lie on the path to the query and the second reading repairs many of the rest. With the extraction frozen the picture changes: the reworded reading leaves 67 of the 229 determined stories looking open, so the executor answers "unsure" on them, while the prompted answerer, working from the same incomplete problem, guesses and is right more often. The executor's advantage on these stories therefore depends on the second reading, and on the determined stories it disappears. Produced by experiments/gen_offtemplate.py and experiments/e2_registry.py.

### B.6 E2 by free roots, depth and remaining latent uncertainty

Accuracy on the 325 E2 items. The one-pass prompt reads and answers in a single call; the prompted answerer and the exact executor answer from the same frozen extraction, with strict scoring, and the margin column is their paired difference. The pipeline, which re-reads on abstention, is the last column, for context. "Undetermined" items are those whose story leaves the query open, where "unsure" is the correct answer and every arm may give it.

How much the observations reveal. The generator was asked for three evidence levels: observations that pin every free root, some, or none. Its check tested whether the observations determined the query, not whether they pinned the roots, so the requested levels do not match the pre-registered definition: of the 106 items generated as fully pinned, only 34 leave a single assignment of the free roots and 32 leave every assignment open. The labels and the pooled results are unaffected. We therefore group by what the observations actually leave, computed with the executor's own enumeration on the generator's problem: one assignment of the free roots (55 items), several but not all (86), or all of them (184; including

every item without observations). The executor's margin over the prompted answerer is

Table 7: E2 accuracy (%) by free roots $R$, causal depth, and how many free-root assignments the observations leave consistent.

| split | $n$ | one-pass | prompted | executor | margin | pipeline |
|---|---|---|---|---|---|---|
| $R$=1 | 117 | 71.8 | 80.3 | 94.9 | +14.5 | 93.2 |
| $R$=2 | 117 | 60.7 | 75.2 | 95.7 | +20.5 | 95.7 |
| $R$=3 | 91 | 52.7 | 67.0 | 97.8 | +30.8 | 98.9 |
| depth ≤5 | 209 | 61.7 | 74.2 | 96.2 | +22.0 | 97.1 |
| depth 6–7 | 104 | 63.5 | 75.0 | 96.2 | +21.2 | 94.2 |
| depth ≥8 | 12 | 66.7 | 83.3 | 91.7 | +8.3 | 83.3 |
| one free-root assignment left | 55 | 76.4 | 76.4 | 98.2 | +21.8 | 100.0 |
| several left | 86 | 57.0 | 64.0 | 97.7 | +33.7 | 97.7 |
| all left open | 184 | 60.9 | 79.3 | 94.6 | +15.2 | 93.5 |
| undetermined ("unsure" is correct) | 96 | 35.4 | 62.5 | 96.9 | +34.4 | 90.6 |
| determined | 229 | 73.8 | 79.9 | 95.6 | +15.7 | 97.8 |
| all | 325 | 62.5 | 74.8 | 96.0 | +21.2 | 95.7 |

larger at more free roots (+14.5, +20.5, +30.8; E3 and E4 do not show this rise, §5.4). By what the observations leave, it is largest where they rule out some assignments but not all (+33.7), so that the answerer must work out which remain, and smallest where they rule out none (+15.2), where many items are undetermined and a prompt can be right by declining. The ≥8 depth row holds 12 items and carries nothing. Of the 96 undetermined stories, the frozen executor reports 93 as open. The remaining ones are read failures: on 2 the extraction is malformed, and on 1 it contains a rule or observation that pins down the free root, so the extracted problem is determined although the story is not. A correct read plus exact execution detects an open query by construction.

### B.7 E3 and E4: protocol and per-$R$ results

Construction. Both sets use one generator (experiments/gen_e3.py; E4 via experiments/gen_grid.py). The free roots are merged pairwise into the query, so the query depends on every one of them; the remaining rules extend a chain to reach the target depth or hang side variables off existing ones, keeping 14 non-root rules at every $R$. Each item's label is computed by the executor on the rendered text and by an independently written brute-force reference, and kept only if they agree. Half the items at each $R$ (E3) or in each cell (E4) are undetermined. E4's design and analysis were committed before its items were generated (PLAN_E4_GRID.md).

Runs. Every answerer receives the generator's own problem. The prompted answerer is Qwen2.5-14B with the same depth-tuned prompt as in E2, capped at 3,840 generated tokens (Appendix B.3); replies that run past the cap are malformed and count as no answer (12 in E3, 7 in E4); counting them as correct instead leaves gaps of +28.9 and +22.2 points. To use both inference slots, a second client answered the $R$=10 items in reverse order while the main run continued; 5 items were answered by both, and the main run's answers are used (the two agree on 4 of them). gpt-oss-120b ran at low and high effort with the 16,384-token budget and the answer parser of Appendix B.2.

### B.8 Additional E1 arms (pre-registered as descriptive, no headline claims)

On a 154-item depth-stratified subsample of the depth sets: CoIn scores 87.7 and the reasoning-tuned same-weights distill 74.7 (median 2,927 thinking tokens; 15 items never reached an answer), vs. the executor pipeline's 95.2 on the full sets—consistent with every earlier finding and registered, per the analysis plan, as descriptive only.

A third descriptive arm fills the remaining cell of §5.2's grid at depth: the depth-tuned prompt simulating from the template parse itself, on the same 120 items (60 per tier).

Table 8: E3 by free roots $R$ (40 problems each; accuracy %; executor time in ms and tokens are medians).

| | executor | Qwen2.5-14B | gpt-oss-120b, low | | gpt-oss-120b, high | | | |
|---|---|---|---|---|---|---|---|---|
| $R$ | ms | acc. | acc. | tokens | acc. | acc. answered | cut off | tokens |
| 1 | 0.20 | 57.5 | 97.5 | 1,366 | 100.0 | 100.0 | 0 | 3,199 |
| 2 | 0.23 | 75.0 | 95.0 | 1,443 | 100.0 | 100.0 | 0 | 3,611 |
| 3 | 0.31 | 65.0 | 100.0 | 1,522 | 92.5 | 100.0 | 3 | 4,553 |
| 4 | 0.46 | 67.5 | 97.5 | 1,808 | 92.5 | 100.0 | 3 | 5,005 |
| 6 | 1.46 | 72.5 | 95.0 | 1,744 | 95.0 | 100.0 | 2 | 5,853 |
| 8 | 5.53 | 62.5 | 95.0 | 1,956 | 92.5 | 100.0 | 3 | 6,068 |
| 10 | 23.04 | 67.5 | 95.0 | 1,934 | 92.5 | 100.0 | 3 | 6,079 |

It answers 60/60 at 8–14 variables and 56/60 at 15–30. On that input the executor is right 120/120 by construction, since an E1 label is what the executor returns on the template parser (all 2,000 labels were re-derived independently by the brute-force checker, §4). The arm therefore measures one thing: how often in-text answering departs from exact execution given identical, perfect input: 4 items in 120, all at the deeper tier (4 vs. 0 discordant, $p$=0.125). The same prompt fed the pipeline's LLM read instead—the matched arm of §5.2—scores 110/120, so at this depth the read, not the medium, carries most of the difference—the same conclusion §5.2 reaches from the LLM-read arms.

### B.9 E1 by depth and by width

Accuracy on the 2,000 pre-registered E1 items (all of the basic question type: one intervention, no stipulated condition), binned by causal depth and by number of variables ($n$ in parentheses). Depth ≤5 occurs only at 8–14 variables, so it is reported here rather than in the grid: on those 787 items the pipeline scores 95.0, the one-pass prompt 89.5 and read-then-simulate 65.7. The executor is flat on both axes over these bins. The 39 items

Table 9: E1 accuracy (%) at depth 6 and ≥7 by number of variables ($n$ in parentheses).

| arm | depth | 8–14 vars | 15–22 vars | 23–30 vars |
|---|---|---|---|---|
| ours | 6 | 97.5 (161) | 95.6 (203) | 98.2 (163) |
| ours | ≥7 | 94.2 (52) | 94.6 (203) | 93.7 (431) |
| one-pass prompt | 6 | 85.1 (161) | 83.7 (203) | 81.6 (163) |
| one-pass prompt | ≥7 | 80.8 (52) | 75.4 (203) | 72.2 (431) |
| read-then-simulate | 6 | 77.0 (161) | 66.0 (203) | 62.6 (163) |
| read-then-simulate | ≥7 | 71.2 (52) | 62.1 (203) | 61.0 (431) |

at depth 10–13 (29/9/1) are included in the ≥7 rows but are too few to plot per depth; on them the executor also drops (to 66.7 at depth 11, $n$=9) and its margin over read-then-simulate narrows to 11–17 points. The one-pass prompt falls with depth at fixed width (89.5→80.8 at 8–14 variables) and with width at fixed depth (85.1→81.6 at depth 6). The read-then-simulate arm (the pipeline's reading followed by chain-of-thought simulation over the re-rendered diagram) is already at 65.7 on the shallowest, narrowest cell and moves with width more than with depth. That arm uses an untuned prompt, and the corrected comparison supersedes it: with the depth-tuned prompt answering from its own reading of the same stories, the prompted answerer scores 91.7 on the 120-item subset and is indistinguishable from the executor (§5.2). The rows below it are reported for the pre-registered arms as run, not as the paper's estimate of what in-text answering can do.

### B.10 Reading and answering on CounterBench alpha

### B.11 Depth on the official split

Binning the official alpha split by parsed causal depth (bins $n$≥105), chain-of-thought falls 69.4→60.8 and CoIn 91.3→84.9 from shallow (≤4) to deep (≥6) stories, while the executor

Table 10: Reading × answering on CounterBench alpha, official key, accuracy (%). $^m$A 300-item sample on which both answerers received the same frozen problem.

| reading | answering | alpha |
|---|---|---|
| template parser | exact executor | 94.5 |
| template parser | prompted answerer | 93.1 |
| template parser | chain-of-thought | 62.9 |
| our reader | exact executor | 92.4 ($93.3^m$) |
| our reader | prompted answerer | $94.3^m$ |

pipeline is flat (90.6→91.8); the benchmark's own difficulty ablation reports the same for CoIn (92→73.5). The one-pass prompt's slide at this moderate depth is small (−1.3); its 15.7-point drop appears only on the deeper, wider depth sets.

### B.12 The official splits by abduction burden

$R$ is the number of exogenous roots the executor enumerates for an item, taken from the gold story with the same expression the executor itself uses: for alpha the roots the intervention leaves free, for backdoor all roots, since those items abduct from the factual world and then intervene. Five alpha items are cyclic and cannot be assigned an $R$. Both arms are the runs of Table 2; the baseline is the one-pass prompt. Every evaluable item is scored against the official key, flagged or not.

Table 11: Official CounterBench splits by free roots $R$ (accuracy %; $b$, $c$: items only the pipeline, or only the one-pass prompt, gets right).

| split | $R$ | $n$ | ours | one-pass | margin | $b$ | $c$ | $p$ |
|---|---|---|---|---|---|---|---|---|
| alpha | 0 | 870 | 92.4 | 88.8 | +3.6 | 59 | 28 | $1\times10^{-3}$ |
| alpha | 1 | 115 | 95.7 | 93.9 | +1.7 | 2 | 0 | 0.50 |
| alpha | 2 | 10 | 100.0 | 80.0 | +20.0 | 2 | 0 | 0.50 |
| alpha | all | 995 | 92.9 | 89.3 | +3.5 | 63 | 28 | $3\times10^{-4}$ |
| backdoor | 1 | 170 | 88.2 | 77.7 | +10.6 | 22 | 4 | $5\times10^{-4}$ |
| backdoor | 2 | 30 | 83.3 | 86.7 | −3.3 | 2 | 3 | 1.00 |
| backdoor | all | 200 | 87.5 | 79.0 | +8.5 | 24 | 7 | $3\times10^{-3}$ |

The split-level contrast is what carries: alpha, where 87% of items need no abduction, against backdoor, where every item does. The per-$R$ cells above $R$=1 hold 10 to 40 items and are too small to read a trend from; the backdoor $R$=2 cell runs the other way on 30 items. Produced by experiments/official_by_R.py from the recorded runs, with no further model calls.

### B.13 Spending more compute on the prompted arm

Two ways of giving the prompted arm more inference compute; neither changes the picture. Self-consistency (Wang et al., 2023) over three sampled chains of the one-pass prompt (≈1.8× our token budget, majority vote) reaches 91.0 on the 300 matched alpha items against the executor's 93.3 (n.s.). A reasoning-tuned distill of our reader's own base weights (DeepSeek-R1-Distill-Qwen-14B, up to 12,288 thinking tokens) lands between chain-of-thought and one-pass prompting overall and still falls on the deepest bins: on the 100-item CB-Hard-XL sample, 32/40 (80%), 24/28 (86%), 13/18 (72%), 6/11 (55%) at depths 6–9, and 3/3 on the sparse depth-10 bin; 78/100 overall.

### B.14 Per-type results, full alpha split (Qwen2.5-14B)

On the full alpha split (250 items per question type: basic, conditional, joint, nested) the pipeline answers 245, 208, 236 and 235 correctly, the one-pass prompt 236, 212, 231 and 215, and chain-of-thought 154, 170, 137 and 188. Conditional is the hardest type for the pipeline and the one-pass prompt, and it holds 35 of the 55 flagged alpha keys (20

forced contradictions, 10 impossible observations, 5 loops); joint and nested hold 10 forced contradictions each, and basic holds none. Official scores are reported as they are; we do not renormalize by the flags.

### B.15 Chunked reading without the fallback

§5.1 gives the full-split figures at 14B (90.8 without the fallback, 92.4 with it, against the one-pass prompt's 89.4). At 7B, chunking alone lifts alpha-sample accuracy from 74.5 to 85.5 and CB-Hard from 42.0 to 75.0, with abstentions falling 41%→8.5% and exact-diagram accuracy rising 3.5%→20.5%.

The single-pass column of Table 12 is the same reader the pipeline's fallback calls, so on CLadder, where the chunked stage abstained on 1,241 of 1,476 items, that column reuses those 1,241 recorded fallback calls verbatim (identical reader, identical item, temperature 0) and calls the model afresh only on the remaining 235; each row records which of the two it came from. Those 1,241 pairs are identical by construction, so the pipeline and the single-pass reader can disagree only on the other 235 items, and only that subset carries a paired test (18 vs. 0 discordant, $p$=$8\times10^{-6}$): 99.6% for chunked reading against 91.9% for a single pass. The alpha column is a fresh run of all 1,000 items.

## C Reading, abstention and cost

### C.1 Reader ablation

Table 12: Reader ablation (accuracy %), and how often the pipeline still abstains after the fallback.

| set (reader) | single | chunked | chunked + fallback | abstained | acc. on answered |
|---|---|---|---|---|---|
| alpha full (14B) | 85.0 | 90.8 | 92.4 | 22 | 94.5 |
| CB-Hard-XL (14B) | 62.5$^{c}$ | 88.0 | 92.0 | 11 | 97.4 |
| CLadder full (14B) | 85.5$^{r}$ | 15.9 | 86.7 | 104 | 93.3 |

Notes $c$ and $r$ are in Appendix D.4; an abstention counts as wrong in the "+ fallback" column. On smaller runs, at 7B chunking lifts a 200-item alpha sample from 74.5 to 85.5 (87.0 with the fallback) and CB-Hard from 42.0 to 75.0 (80.0); at 14B, chunked reading scores 95.5 on CB-Hard (96.5 with the fallback) and 87.5 on backdoor, where the pipeline abstains on 5 items and is 89.7% accurate on the rest.

### C.2 Whole-problem extraction accuracy

A recovered rule set is not a recovered problem: an extraction can carry every mechanism and still misplace the evidence, the intervention, the query variable or its polarity, and the executor will then answer a different question exactly. The runner records all five components per item. On E2 ($n$=325) the reader recovers the evidence, the intervention, the query variable and its polarity on every item, and the mechanisms on 82.8% (269/325), so whole-problem accuracy equals mechanism accuracy here at 82.8% and no item is lost to the other four. That is a property of these items, not a general one: the components are reported separately precisely because they can come apart, and on the single-pass alpha arm they do (Appendix C.3). Produced by experiments/extraction_fidelity.py.

### C.3 Abstention as selective prediction

The last two columns of Table 12 treat the pipeline's abstention as selective prediction (El-Yaniv and Wiener, 2010): the number of items still unanswered after the fallback, and accuracy on the items it does answer. Before the fallback the chunked stage abstained on 40 alpha items; the single-pass retry resolved 18, of which 16 were correct (CB-Hard-XL: 20, 9 resolved, 8 correct; CLadder: 1,241, 1,137 resolved, 1,046 correct). Of alpha's 22 remaining

abstentions, 15 fall on audit-flagged items and 7 on items whose extraction was wrong. Abstention never fires on a correct read of a sound story (0 of the 888 exactly matched well-posed items); it does fire on 13 correct reads of defective ones, which is the behaviour we want. As a detector of reading failure it is conservative: of the 57 structurally wrong extractions among alpha's 945 well-posed items (63 over the whole split), 7 abstained, 50 answered, 14 of those wrong; on CB-Hard-XL 87 were wrong, 11 abstained, 76 answered, 5 wrong. The stored self-consistency artifact keeps only the majority vote, so a chain-disagreement abstention baseline could not be reconstructed without new runs; on the 300 matched items, majority vote answers everything at 91.0 while the pipeline answers 97.7% at 95.6 on those it answers.

Matching mechanisms is not the same as reading the whole problem. Across the pipeline's six full-split runs (2,483 exactly matched mechanism sets) every disagreement with the key is audit-flagged, but in the single-pass arm on alpha two of its 789 exactly matched items still answer wrongly on unflagged items—items 643 and 61, where the mechanisms are right and the intervention or the stipulated condition is not. This is why we report whole-problem parse correctness separately from mechanism match (§5.2).

### C.4 Executor states, and what the retry costs

build_status returns the state alongside the answer and agrees with the shipped builder on all 325 abduction items. Their states: 221 determined, 102 underdetermined, 2 invalid. Of the 102 underdetermined, 93 are items whose story genuinely leaves the query open (so declining is correct) and 9 are stories that do determine an answer, i.e. read errors.

The retry rule we froze fires on any no-answer state, so it fired on 95 of the 96 undetermined items. In 8 of those it returned a confident yes or no, and all 8 were wrong. Restricting the retry to the inconsistent and invalid states—the two that indicate a failed read—scores 96.6% on these items against the 95.7% we report. That figure is retrospective, computed from the recorded runs of the frozen policy: it diagnoses the policy rather than replacing it, and a revised policy would carry its own version and its own headline numbers. The frozen policy is the source of every result in this paper. On alpha the frozen rule is nearly harmless: the retry fired on 40 items, resolved 18, and only 2 of those were wrong, none of them audit-flagged.

### C.5 Compute accounting

Median generated tokens per item (matched samples of 48 alpha / 50 XL items, same prompts as the main runs): on alpha, ours 441 vs. 241 for the one-pass prompt and 320 for chain-of-thought; on CB-Hard-XL, ours 1,597 vs. 721 and 574. Fuller accounting (calls per item are exact from the pipeline's control flow, $1 + \lceil \text{statements}/3 \rceil + 1$, plus one on fallback; prompt tokens are estimated as characters/4 on the exact prompt strings, since the serving stack did not log them; wall-clock is the same matched-sample stopwatch as the token medians): Prompt-token figures are per item, summed over that item's calls. Per call

Table 13: Cost per item (p50 = median, p90/p95 = percentiles; CoT = chain-of-thought).

| set | method | calls p50/p90 | out tok. p50 | prompt tok. p50 (est.) | s/item p50/p95 |
|---|---|---|---|---|---|
| alpha | ours | 4/5 | 441 | 3,412 | 29.5/40.5 |
| alpha | one-pass | 1/1 | 241 | 656 | 15.6/24.7 |
| alpha | CoT | 1/1 | 320 | 90 | 21.2/38.3 |
| CB-Hard-XL | ours | 10/12 | 1,597 | 10,085 | 98.6/141.3 |
| CB-Hard-XL | one-pass | 1/1 | 721 | 766 | 44.0/67.8 |
| CB-Hard-XL | CoT | 1/1 | 574 | 200 | 40.2/57.6 |

the largest prompt we send is far smaller: over 40 items of each set, the biggest single call is about 1,100 estimated tokens on alpha and 1,270 on CB-Hard-XL, against the reader's 4,096-token context, which is the basis for the claim in §4 that no call approaches its limit. Hardware: one Apple M1 Max laptop (64 GB), Ollama, 4-bit quantised weights. The chunk

calls are independent and could run in parallel; we ran them sequentially. Prompt cost is dominated by re-sending the instruction block once per call.

### C.6 Enumeration cost

Two counts must be kept apart. $R$ is the number of free roots, those the intervention does not set. The executor enumerates $2^{U_e}$ worlds, where $U_e$ is the number of roots it must enumerate: every root (one more than $R$ when the intervened variable is itself a root, as in our generated sets) when the observations describe the actual world, and only the free roots ($U_e = R$) when they describe the changed world or there are none. Across CounterBench, CLadder, E1 and E2, $R$ never exceeds 3: alpha median 0—on most alpha items there is nothing to abduct, which is why the prompted answerer matches the executor there (§5.2) while the confounded and abduction sets separate them—maximum 2; backdoor median 1, maximum 2; CB-Hard, CB-Hard-XL and both depth sets $R$=0 throughout (one root, the intervened one); CLadder median 1, maximum 2; E2, which is built to vary this quantity, $R \in \{1, 2, 3\}$ by construction. Measured executor time on the template parser is below 0.03 ms per item on every dataset, and below 40 ms in total for all datasets combined. E3 raises $R$ to 10 (1,024 worlds), where the executor takes a median of 23.0 ms per problem (Appendix B.7).

Table 14: Latent state in each generated set: free roots $R$, roots the executor enumerates ($U_e$), and the free-root assignments still consistent after conditioning on the observations (median, and maximum).

| set | observations about | free roots $R$ | enumerated $U_e$ | consistent assignments | determined |
|---|---|---|---|---|---|
| E1 | none | 0 | 0 | 1 (max 1) | 100% |
| E2 | actual world (66%) | 1–3 | 1–4 | 2 (max 8) | 70% |
| E3 | actual world (64%) | 1–10 | 1–11 | 6 (max 1,024) | 50% |
| E4 | actual world (61%) | 1–6 | 1–7 | 4 (max 64) | 50% |

In E2–E4 a problem either has observations, always about the actual world, or has none (34–39% of problems), in which case every assignment of the free roots stays open. The number of consistent assignments can be as large as $2^R$, so $R$ bounds the remaining uncertainty rather than fixing it; E2 varies that uncertainty directly through its observations (Table 7 groups items by what they leave). Enumeration doubles with every free root, so latent spaces much larger than ten free roots would call for knowledge compilation or weighted model counting in its place; we have not tested beyond $R$=10.

### C.7 A chunked read, end to end

For CB-Hard-XL item H0 (22 variables, 20 causal statements), the splitter call returns the statements verbatim; seven chunk calls of three statements each return mechanism lists; the merge keeps the first translation of each effect; the question call reads the intervention. The executor enumerates the unobserved roots, applies the intervention, and reports the unanimous query value—correctly. Measured generated tokens for this item: 1,517.

### C.8 One item end to end

> Worked example (real item, abridged).
>
> Story: "Ziklo causes not Blaf. Blaf and Trune together cause Vork. We observed Trune." Question: "Would Vork occur if not Ziklo instead of Ziklo?"
>
> Read (constrained JSON, shown as rules): Blaf = NOT Ziklo; Vork = Blaf AND Trune; observed: Trune on; change: Ziklo off; asked: Vork. (With a single parent the two ways of writing the "not" coincide—negating the effect and negating the cause both give Blaf = NOT Ziklo—so what the schema rule forbids is setting both flags for one "not", which cancels. Scope does matter once a rule has more than one parent: NOT ($A$ AND $B$) and $A$ AND NOT $B$ differ on half their inputs.)
>
> Execute: roots are Ziklo and Trune. Observation keeps only worlds with Trune on. Apply the change (Ziklo off): Blaf = on (Ziklo is the only thing that switches Blaf off, and Ziklo is off), Vork = Blaf AND Trune = on. Every surviving world agrees ⇒ yes. (Had the worlds disagreed, the executor would return "unsure" and the pipeline would re-read once with the single-pass reader.)

## D Executor validation and the benchmark audit

### D.1 Executor tests against independent references

The exactness claim rests on the executor being right, so we test it against two references written separately from it: a brute-force enumerator over all variable assignments (it keeps the assignments that satisfy every mechanism, and applies abduction–action–prediction by definition), and a fixpoint propagator that solves the equations by repeated relaxation rather than in topological order. On 5,000 random acyclic boolean models (3–9 variables, random operators, negations, observations, interventions and queries), across both conditioning worlds, all 10,000 comparisons agree, and the two references agree with each other. The references are independent on the point that matters most here: they enumerate all roots, including any the question intervenes on, while the executor excludes intervened roots in the changed-world branch, so agreement is evidence about that rule rather than a shared assumption. These are implementation tests, not a proof.

Writing the regressions found one case where executor and reference must disagree, and it changed the pipeline. If an extraction contains two rules for the same effect, the reference treats both as constraints while a lookup table keeps one, so executing such an extraction silently picks a reading. The executor now records the conflict and the pipeline refuses the item as an unresolved extraction rather than answering it. No number in this paper moves: the chunked merge dedupes upstream, and no stored extraction on any reported run contains a conflicting rule. The regression suite also covers factual versus changed-world evidence, an empty set of consistent worlds, genuine ambiguity, negation on either side of a rule, two simultaneous interventions, and query polarity.

One case is worth stating because it is hand-checkable and because getting it wrong would silently manufacture answers. Let $X$ and $U$ be roots with $Y = X \vee U$; observe $X$ and $Y$ factually, and ask whether $Y$ would hold under do($X$=0). The evidence does not pin $U$, so the counterfactual is undetermined; substituting the intervention into the factual world would wrongly force $U$=1 and answer "yes". Both semantics decline, for the two different reasons the executor distinguishes (Appendix C.4): abducting from the natural world leaves both $U$ values surviving and disagreeing, while checking the evidence in the changed world finds no surviving world at all. This is a regression test in the released script. On the benchmark it is moot: basic, joint and nested items carry no observations, so only conditional and backdoor items can exercise it.

### D.2 Flagged answer keys: four worked examples

Item 394 (nested type; both clauses are interventions under the benchmark's definition)—story: "Glent causes Razz and Pex, Razz causes not Zurn, Pex or Zurn causes Melf, Melf causes Zlim"; question: "Assume not Glent, and further suppose not Zurn. Would Zlim occur?" No Glent means no Pex (rule 1); Zurn is assumed off; Melf needs Pex or Zurn (rule

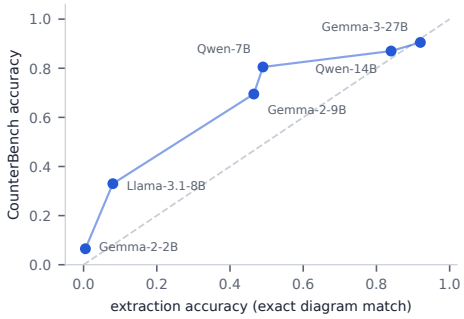


Figure 3: Final accuracy rises with reading accuracy across six open readers (alpha, single-pass; dashed line $y{=}x$).

3), both off; Zlim needs Melf (rule 4). The story forces "no" in four quoted steps; the official answer is "yes." Item 544: "Praf causes Vank, Vank causes Scud, Scud causes not Wrenk, Wrenk or Vank causes Yobb, Yobb causes Glim, Glim causes Klep." Question: "Would Klep occur if not Praf and not Wrenk?" No Praf means no Vank; Wrenk is assumed off; Yobb needs Wrenk or Vank, both off; so no Glim, no Klep: the story forces "no"; the key says "yes." Item 70: "Ziklo causes Blaf, Blaf causes Trune, Trune or not Blaf causes Vork, Vork causes Sline and Frim, Frim causes Qado, Qado and Sline together cause Lumbo. We observed Vork." Question: "Would Lumbo occur if not Ziklo?" Without Ziklo: no Blaf, no Trune, but "Trune or not Blaf" is then true, so Vork holds (consistent with the observation), hence Sline, Frim, Qado, and therefore Lumbo: forced "yes"; the key says "no." Every flag is recomputable from the story text by the released script. Backdoor item 13: "Blaf causes Ziklo, Blaf and Trune together cause not Vork, Vork causes Sline, and Sline causes Lumbo. Blaf∼Bern(0.4). We observed not Vork." Question: "Would Lumbo occur if not Ziklo instead of Ziklo?" Observing not Vork forces Blaf and Trune both on (the only way the AND-cause of "not Vork" holds), so Vork is off, so Sline is off, so Lumbo is off; Ziklo feeds nothing downstream, so changing it changes nothing: the story forces "no"; the key says "yes." Moving the "not" onto Trune instead leaves three of the four root assignments consistent with the observation, and all three still force "no", so no reading rescues the key.

### D.3 Audit semantics and a second-source cross-check

Every flag is computed under the question-type definitions CounterBench itself gives: basic $Y_x(u)$; joint $Y_{x,z}(u)$ ("if not X and not Z"); nested $Y_{Z_x}(u)$, phrased "Assume not X, and further suppose not Z", where the benchmark defines "further suppose" as an additional intervention; conditional $Y_x(u) \mid Z_x(u){=}z$, the condition holding in the changed world; backdoor, observations of the actual world with an unobserved common cause. Mechanisms are read as stated: "A causes not B" sets $B{:=}\neg A$; "A and B together cause C" sets $C{:=}A \wedge B$; "A or B causes C" sets $C{:=}A \vee B$. As a check that the flags are not an artifact of our story-text parser, we rebuilt every alpha and backdoor model from CounterBench's separately released meta_model description of each graph (different sentence templates), using a separately written parser, and executed it with the same semantics: all 80 flagged items, and no others, are flagged again, with identical structure on all 1,200 items. Both sources are text, so this is redundancy across two independent renderings and two parsers rather than an executable specification from the benchmark authors, which CounterBench does not ship. The flags are also insensitive to the two semantic choices a reader might contest. First, evaluating the conditional-type condition in the factual world instead of the changed world leaves 30 of the 35 conditional flags in place (all 20 forced contradictions and all 5 loops), resolves 5 of the 10 impossible-observation flags, creates no new flag, and moves the oracle from 94.5 to 95.0. Second, reading the second clause of a joint or nested question ("and further suppose not Z") as an observation rather than a second intervention leaves all 80 flags in place: the 20 joint and nested items become unsatisfiable rather than forced contradictions—the stated facts then have no consistent world at all—and no flag resolves. Items 394 and 544 above illustrate the difference: under the observation reading both are

unsatisfiable, and the official key is recovered only by dropping the second clause outright, which discards a stated fact rather than interpreting it.

### D.4 Notes for Tables 2 and 12

Table 2 covers the full official splits (alpha $n$=1,000, backdoor $n$=200), all rows run in one harness at one model size. The gold-parse reference is the executor on the template parse; with disputed keys it is a reference point, not a mathematical upper bound on official-key accuracy. In Table 12 the alpha and CLadder rows are full splits, the 7B row is a 200-item sample, $^{c}$ marks the 16k-token context, "—" means not run, and $^{r}$ marks the CLadder single-pass column, which reuses the pipeline's own recorded fallback calls on 1,241 of 1,476 items (Appendix B.15).