CONSISTENT BUT WRONG: ANALYZING WHAT LLM REASONING LACKS VIA STATE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models are increasingly used for multi-step reasoning. They decide at the token level, and every later step builds on what they have written. Sampling-and-selection (best-of-N, majority voting) chooses among chains already written. A *consistent error* occurs when all sampled chains that reach a decisive step write the same wrong successor state, so the pool holds no correct candidate at that step. We use *state decoding* to analyze what the model's reasoning lacks at such a step. Its *support* supplies the candidate states at each step, the *scoring* ranks them, and the *search* assembles the decisions. An enumerated support contains the correct successor whatever probability the model assigns to it, and with an exact search every remaining error is a scoring error. With Qwen3-4B, we run SAVI (State-Aware Viterbi Inference), which enumerates the candidates and searches exactly on *MuSR* (belief tracking) and *Belief-R* (belief revision). In both tasks, most failures that more samples do not repair are consistent errors. Once the candidates contain the correct state, the model's likelihood picks a belief table on MuSR by the sentence that asserts the move, not by who saw it. On Belief-R it picks a reading by its wording. When the prompt states the correct reading, the model answers correctly, so it fails at choosing the reading. For Qwen3-4B, only signals outside the likelihood pick the correct candidate above chance. Of 19 MuSR items on which voting stays wrong, a verifier built from the annotated facts fixes 15 when it scores enumerated states and 3 when it scores the sampled pool. On *Countdown* the chains disagree, and frequency does not track target reachability: of 45 items on which every sampled chain fails, state decoding solves 9 with frequency scoring and 44 with an exact verifier. With uniform proposals and the same verifier, it solves 39 to 42. Separating the candidates without labels or an exact verifier remains open.

1 INTRODUCTION

A single wrong step can decide the answer. Language models are increasingly used for multi-step reasoning, in which they write intermediate steps before the answer (Wei et al., 2022). They decide at

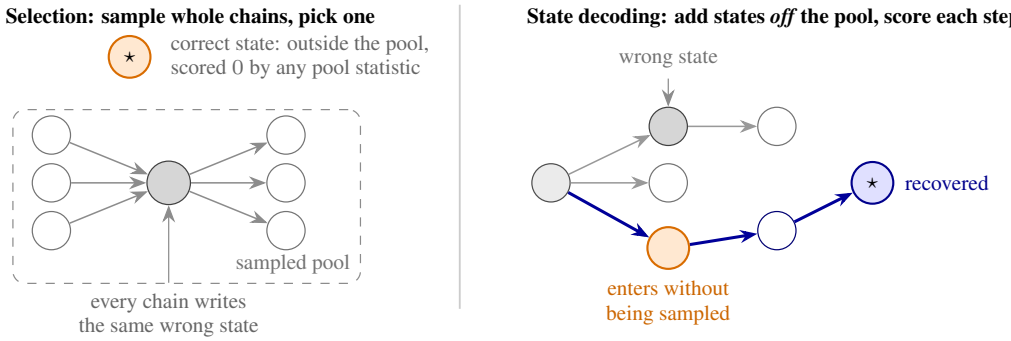


Figure 1: **Where candidates come from.** The belief vignette at the decisive step. *Left:* a consistent error (Section 3.2). *Right:* the decoder takes the blue path only if the correct state scores higher.

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.

the token level, and every later step builds on what they have written. A wrong step can overturn an interpretation that the model wrote earlier. A model that reads the bank of a fishing trip as a river bank then answers as if it were a financial bank (Mozer et al., 2026a). *MuSR* measures the same overturning at the level of belief: *Ann places her keys in a drawer and leaves; Ben moves them to a box; where will Ann look?* The model follows the keys rather than Ann's belief. In both cases the model writes the wrong interpretation at one step. We ask what the reasoning lacks at such a step: the correct interpretation as a candidate, or a score that ranks it above the wrong one.

Why sampling-and-selection is not enough. Repeated sampling with selection (best-of-N, majority voting) can return only an answer that some sampled chain gives. The number of samples needed before some chain gives the correct answer grows as the model's probability on that answer shrinks (Chen et al., 2026). We call a step *decisive* if writing a different result at that step would change the final answer. A *consistent error* occurs when every sampled chain that reaches a decisive step writes the same wrong result there (Figure 1, left). The pool then lacks the correct result at that step, and more samples tend to repeat the error.

What state decoding separates. *State decoding* moves the decision from the token to the state: a derivation becomes a sequence of states, and the decoder chooses among candidate states at each step. A decoder has three components (Section 3): the *support* supplies the candidates, the *scoring* ranks them, and the *search* assembles the decisions at successive steps into one trajectory. We run SAVI (State-Aware Viterbi Inference), a state decoder that searches with the Viterbi algorithm (Viterbi, 1967). When every state is kept, this search returns the highest-scoring trajectory exactly, so a wrong answer comes from the support or the scoring. *Countdown* (Section 4.2) is an arithmetic task with an exact verifier, and we vary the scoring over its resampled candidates. For *MuSR* (Section 4.3) and *Belief-R* (belief revision, Section 4.4) we enumerate the candidates, so they include the correct state.

Summary of contributions.

- We define and measure the consistent error, which leaves the sampled pool without a correct candidate at a decisive step (Section 3.2). Most failures that more samples do not repair on MuSR and Belief-R are consistent errors (Sections 4.3 and 4.4). Before any decoder is chosen, a four-term split of each item shows whether the pool lacks the correct answer (Section 3.1).
- We use state decoding to attribute a failure to one component of the decoder (Section 3): when the candidates contain the correct state and the search is exact, every remaining error is a scoring error.
- The missing component differs by task. On Countdown the scoring fails. On MuSR and Belief-R the sampled chains lack the correct state, and the model's likelihood does not pick it even when enumeration supplies it. With Qwen3-4B, only signals outside the likelihood pick the correct candidate above chance. The search adds no measurable gain on Countdown (Sections 4.2 to 4.4).

2 PRELIMINARIES

Given a problem x, a language model π_θ generates a **reasoning chain** (Wei et al., 2022), a token sequence $w_{1:M}$ that ends in a final answer. Generation is autoregressive, so $\pi_\theta(w_{1:M} \mid x) = \prod_{i=1}^{M} \pi_\theta(w_i \mid x, w_{<i})$. The greedy chain is the one with the argmax token at each position, and y_{greedy} is its answer.

Selection baselines draw N chains i.i.d. from π_θ at temperature 1 (on Countdown with the model's default top-k=20 and top-p=0.95 truncation) and choose among the multiset of their answers $\mathcal{Y}_N = \{y_1, \ldots, y_N\}$. We call the N chains the **sampled pool**. Majority voting, or self-consistency (Wang et al., 2023), returns the most frequent answer y_{mode} in $\mathcal{Y}_N$ (Figure 2(b)), and ties are scored as incorrect. A verified best-of-N applies an exact check to each answer in $\mathcal{Y}_N$, so it is correct on an item with gold answer y^* exactly when $\max_{y \in \mathcal{Y}_N} \mathbb{I}[y{=}y^*] = 1$, where $\mathbb{I}(\cdot)$ is the indicator function.

A **canonical state** S records what the chain has established so far, in the form the next step acts on (Section 4). *Countdown* requires an arithmetic expression that uses k given numbers and evaluates to a target. Each of the $k-1$ steps combines two remaining numbers into one. A Countdown state S is the sorted multiset of values still available, together with the target (Figure 2). A derivation is then a state trajectory $\tau = (S_0, \ldots, S_T)$. The index t counts derivation steps, each realized by a multi-token segment, so a chain traces one trajectory.

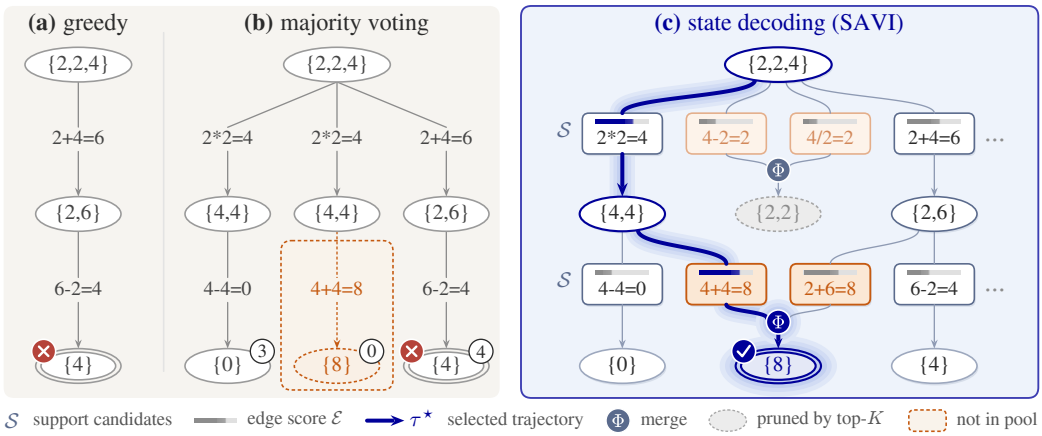


Figure 2: **Three decoders on one Countdown item** with numbers $\{2, 2, 4\}$ and target 8. *(b)* Circled numbers are the votes of the $N{=}7$ chains in the sampled pool. Every chain that reaches $\{4, 4\}$ writes $4{-}4{=}0$. *(c)* One pass of Algorithm 1 with $K{=}2$. Score bars are illustrative.

3 Consistent Errors and State Decoding

3.1 The four-term split

The four-term split measures how much of the headroom above the greedy chain a rule over the sampled pool can recover. With y_{greedy}, $\mathcal{Y}_N$ and y_{mode} as in Section 2, every item with gold answer y^* satisfies

$$1 = \underbrace{\mathbb{I}[y_{\text{greedy}}{=}y^*]}_{\text{greedy}} + \underbrace{\mathbb{I}[y_{\text{mode}}{=}y^*] - \mathbb{I}[y_{\text{greedy}}{=}y^*]}_{\Delta_{\text{vote}}} + \underbrace{\max_{y\in\mathcal{Y}_N} \mathbb{I}[y{=}y^*] - \mathbb{I}[y_{\text{mode}}{=}y^*]}_{\Delta_{\text{select}}} + \underbrace{1 - \max_{y\in\mathcal{Y}_N} \mathbb{I}[y{=}y^*]}_{\epsilon_{\text{miss}}}.$$

Averaged over items, $\Delta_{\text{vote}} = \text{mode} - \text{greedy}$, $\Delta_{\text{select}} = \text{any} - \text{mode}$ and $\epsilon_{\text{miss}} = 1 - \text{any}$, where greedy, mode and any are the accuracies of the greedy chain, the majority vote and verified best-of-N. Δ_{select} is the share of items whose pool holds a correct answer that the vote does not return. No rule over the pool can improve on the vote by more than Δ_{select}. ϵ_{miss} is the share of items on which no sample at budget N is correct. At a fixed budget, only candidates from outside the pool can reduce ϵ_{miss}. All four terms are computed on the answers of the greedy chain and the sampled pool, before any decoder is chosen. Where the answer determines the derivation, an item counted in ϵ_{miss} also shows that the pool lacks a correct derivation.

3.2 Where an error occurs

In the sampled pool, the candidates at a step are the successor states that the N sampled chains write at that step. Each candidate is ranked by its frequency, the number of chains that wrote it. Ranking by frequency is majority voting applied to each step. Decoding can then fail at a step in two ways: the correct successor is not a candidate, or it is a candidate ranked below a wrong one. The first is a failure of the *support*, which determines what the candidates are and where they come from. The second is a failure of the *scoring*, which ranks the candidates.

We call a step *decisive* if the final answer depends on which successor is written there, provided that the derivation continues correctly from that successor. Errors at a decisive step fall into two cases, depending on whether the sampled chains that reach the step agree on its successor. When the chains disagree, a larger budget may add the correct successor to the pool. A *consistent error* occurs when all sampled chains that reach a decisive step write the same wrong successor there (Figure 1). The correct successor is then not a candidate, so changing the scoring or the search (Section 3.3) over the same pool cannot select it. Repairing the error at that step requires a change of support. The agreement of

	E1 frequency	E2 likelihood	E3 verifier-weighted
S1 sampled chains	majority voting	—	verifier-weighted rerank
S2 resampled	state decoding, λ=0	—	state decoding, λ>0
S3 enumerated	—	state decoding, λ=0	state decoding, λ>0

Table 1: **Support × Scoring.** *Countdown* uses S_2, *MuSR* and *Belief-R* use S_3, and baselines use S_1.

Algorithm 1 SAVI(S_0, $\mathcal{S}$, $\mathcal{E}$, Φ, K, T)

Require: initial state S_0, support $\mathcal{S}$, edge scorer $\mathcal{E}$, canonicalizer Φ, beam K, horizon T (per item)
$L_0 \leftarrow \{S_0\}$; $\text{score}_0(S_0) \leftarrow 0$
for $t = 0, \ldots, T-1$ **do**
  $C_t \leftarrow \mathcal{S}(L_t)$ ▷ support: candidates $S \to S'$
  canonicalize successors by Φ, keep back-pointers
  $\text{score}_{t+1}(S') \leftarrow -\infty$ for each successor S' in C_t
  for each canonical edge $S \to S'$ in C_t **do**
    $\text{score}_{t+1}(S') \leftarrow \max\{\text{score}_{t+1}(S'),\ \text{score}_t(S) + \mathcal{E}(S \to S')\}$ ▷ scoring
  $L_{t+1} \leftarrow$ top-K of layer $t{+}1$ by score_{t+1}
return answer read off the backtracked argmax $\tau^\star$

the chains suggests that the model assigns the correct successor low probability, so drawing more chains and resampling at that step both tend to repeat the wrong successor. Enumeration includes the correct successor by construction, whatever probability the model assigns to it.

If the candidates include the correct successor and the search is exact (Section 3.4), every remaining error is a scoring error. Candidates that the model did not sample have zero frequency, so the scoring must use the model's teacher-forced likelihood or the score of an external verifier. The teacher-forced likelihood comes from the same model that wrote the consistent error.

3.3 STATE DECODING

A decoder has a third component besides its support and scoring: the *search*, which assembles the decisions made at successive steps into one trajectory. Selection over sampled chains has no search. It takes whole chains as candidates and decides once, at the end. *State decoding* instead decides among candidate states at each step. The support takes one of three forms: the sampled chains (S_1), successors resampled at each node (S_2), or successors enumerated at each node (S_3). Scoring uses frequency (E_1), the model's teacher-forced likelihood (E_2), or either score weighted by an external verifier (E_3). SAVI is the state decoder that we run. Within a task, every run of SAVI uses the same search, the Viterbi recursion of Section 3.4, so two runs differ only in their support or scoring. In Section 4, state decoding refers to SAVI. Other decoders can be described by the same three components (Section 5). Table 1 classifies each decoder that we measure by its support and scoring.

3.4 SAVI: VITERBI DECODING OVER A STATE TRELLIS

The candidates form a *trellis*. Layer t holds the states kept at step t, and edges join each kept state to its candidate successors in layer $t{+}1$ (Figure 2(c)). SAVI decodes by maximum-a-posteriori inference over the state trajectories τ of Section 2:

$$\tau^\star = \arg\max_{\tau} \sum_{t} \big[\log P_{\text{model}}(S_{t+1} \mid S_t) + \lambda \log V(S_t \to S_{t+1})\big], \tag{1}$$

where P_{model} is the model's probability of the successor given the current state, and the verifier V estimates whether the step still leads to the gold answer. Frequency scoring (E_1) estimates P_{model} by the share of steps sampled at S_t that move to S_{t+1}. Likelihood scoring (E_2) replaces P_{model} with the teacher-forced probability of the successor's text in one fixed template. This probability covers one wording of the successor, not every wording of that state. $\lambda \geq 0$ weights the verifier against the model ($\lambda = 0$ uses the model alone). Each summand is the edge scorer $\mathcal{E}$ of Algorithm 1.

We search with the Viterbi recursion (Viterbi, 1967) because it is exact and avoids enumerating the trajectories, whose number grows exponentially with the number of steps. When every state is kept,

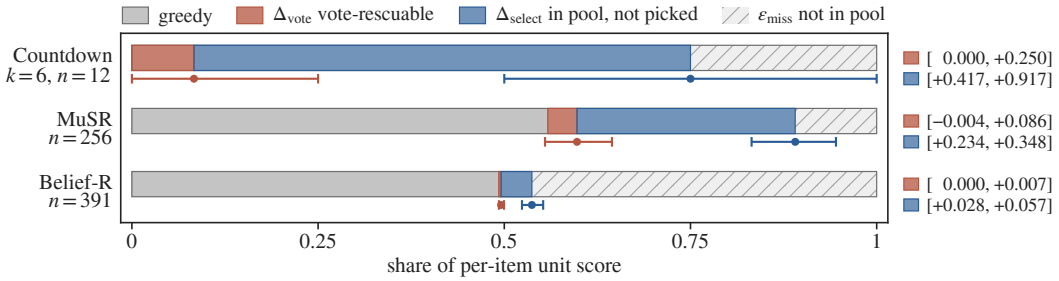


Figure 3: **The split of Section 3.1 on three tasks.** Each bar sums to 1. The two lines under each bar are 95% paired-bootstrap intervals for Δ_{vote} and Δ_{select}, with endpoints listed on the right. The *Belief-R* row is read on BREU (Section 4.4). The budget is $N{=}64$ on *MuSR* and $N{=}32$ on *Belief-R*. On *Countdown* $k{=}6$, mode is read on the first 8 of the 40 to 74 chains sampled for each item and any on all of them.

the recursion returns the trajectory that maximizes Equation (1), so a wrong $\tau^\star$ comes from the support or the scoring, not from the search. The model samples or scores each step without the earlier steps in its input (Appendix A.1), so each edge score depends only on the two states that the edge joins. Among the trajectories that reach a state, only the best can then be a prefix of $\tau^\star$. Layer by layer, the recursion keeps that best score at each state with a back-pointer to its predecessor, and then follows the back-pointers from the best terminal state (Algorithm 1, Figure 2(c)). The search keeps every state on MuSR and Belief-R but only $K{=}8$ per layer on Countdown, where it is approximate.

4 Experiments

Numbers are marked ORACLE when they depend on an exact verifier, the facts that MuSR's generator provides with each item, or a probe fitted on human labels. Each supplies one component, so the other components are measured while that one is correct.

4.1 Where the headroom sits

The gain of majority voting over greedy decoding is indistinguishable from zero on all three tasks. Δ_{vote} is $+0.083$ on Countdown $k{=}6$, $+0.039$ on MuSR and $+0.003$ on Belief-R (Figure 3). The headroom above the greedy chain therefore lies in Δ_{select} and ϵ_{miss}. Δ_{select} is $+0.667$ on Countdown, $+0.293$ on MuSR and $+0.041$ on Belief-R, and none of its intervals contains 0. Sections 4.2 and 4.3 study the part of this headroom that each task's selection baseline does not recover. On Countdown the selection baseline is verified best-of-N, which returns a correct answer whenever the pool holds one, so this part consists of the items that no sampled chain solves. MuSR has no such check, so its baseline is majority voting, and this part also includes items whose pool holds the correct answer.

A vote can differ from the greedy answer only where the samples disagree with it. MuSR samples mostly agree with the greedy answer: their most frequent answer is the greedy answer on 208 of 256 items, including 78 of the 113 items on which greedy decoding is wrong. The same holds on Belief-R, where both keep the conclusion on nearly every item (Section 4.4). Consistent errors (Section 3.2) would produce such agreement on a wrong answer. No such agreement appears on Countdown: at $k{=}6$, the eight chains that the vote counts end in eight different states on 9 of 12 items.

4.2 Countdown: frequency does not track target reachability

On Countdown, the resampled support (S_2) holds a path to the target on nearly every item. The runs of state decoding below share this support and the search (Section 3.3), so they differ only in the scoring. We vary the scoring through the verifier V of Equation (1). Most sampled chains make the target unreachable before the finished chain is verified. With one step left, 0.09, 0.02 and 0.03 of the chains in a verified best-of-N pool can still reach the target at $k{=}4, 5, 6$, averaged over items (Figure 4(a)). No later step can undo a step that makes the target unreachable.

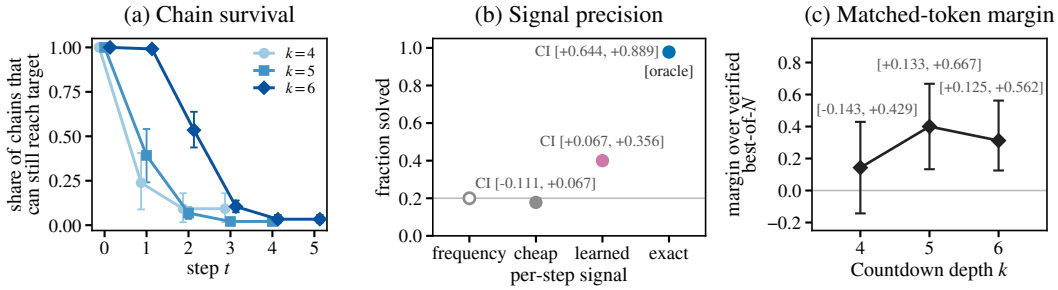


Figure 4: **Countdown.** *(a)* Averaged over the 33 items of (c) whose best-of-N chains are stored (n=10, 11, 12). *(b)* The 45 items at k=6; the cheap signal is a smaller network on a subset of the learned head's features. For both learned signals, λ=0.5 is run first and λ=1 only on the items that it leaves unsolved. *(c)* State decoding with the exact verifier at matched tokens (Appendix A.2), on solvable items where a heuristic taking the move closest to the target fails (n=14, 15, 16). Error bars are 95% bootstrap intervals, paired in (c) and clustered by item in (a); CI in (b) is paired against frequency alone.

State decoding solves few items with frequency and nearly all with the exact verifier. The test items are 45 solvable k=6 items on which none of 64 chains sampled from Qwen3-4B with a LoRA adapter is correct. State decoding with frequency alone (λ=0) solves 0.200, and every solution it finds uses a step that none of these 64 chains took (Appendix B.2). A learned solvability head raises the solved fraction to 0.400 (Figure 4(b)), where an item counts as solved if either of two weights λ solves it. The exact verifier rejects every step from which the target is unreachable [ORACLE], and state decoding with it solves 0.978, or 44 of the 45 items. With uniform proposals in place of the model's, state decoding with this verifier still solves 39 to 42 (Appendix B.2), so most of these solutions do not depend on the model's proposals.

State decoding with the exact verifier outperforms a token-matched verified best-of-N at k=5 and 6. The margin of state decoding over the verified best-of-N is +0.143, +0.400 and +0.312 at k=4, 5, 6 (Figure 4(c)). A separately trained feasibility classifier in place of the exact verifier gives a negative margin at k=6 (Appendix A.2). Besides where the verifier is applied, the two decoders differ in support and search. The gain from the search is indistinguishable from zero. A *masked best-of-N* draws 8 candidate successors at each step of a chain and keeps those that the exact verifier accepts. It then follows one of them, chosen with probability proportional to its count among the 8. The beam, the merge and the Viterbi recursion of Algorithm 1 are omitted. Its runs draw on four sampling seeds that state decoding does not use. State decoding minus masked best-of-N averages −0.18 [−0.50, +0.10], +0.07 [−0.05, +0.20] and +0.06 [0.00, +0.13] at k=4, 5, 6.

4.3 MuSR: BELIEF UPDATES DO NOT FOLLOW WHO SAW THE MOVE

MuSR `object_placements` (Sprague et al., 2024) contains stories like the belief vignette of Figure 1: objects are moved, and each move may be seen by any subset of the other characters. A canonical state is the belief table that records where each character believes each object is (Table 4). We use 19 *certified* items. On each, the majority answer stays wrong at 1024 samples and turns right once the annotated facts are given (Appendix C.1). MuSR's generator provides these facts with each item: who moved each object and who saw each move. When the model first extracts these facts from the story itself, the answers stay wrong (F3 in Figure 7). The failure therefore lies in this extraction.

The model writes the wrong table at forks and where the answer is set. A *fork* is a step at which more than one next table is possible, depending on who saw the move. The annotated paths of the certified items contain 30 forks. For each fork, the model receives the annotated table of the previous step and the *annotated sentence*, which names the moved object and its new location but not who saw the move. The model's most frequent table is wrong at 80% of these forks. Every earlier step is given correctly, so these errors arise at the fork itself. In 18 of the 24 wrong tables, the object moves in every character's belief, including the beliefs of characters who did not see the move. The model's count of wrong forks is indistinguishable from those of two model-free rules: moving the object for

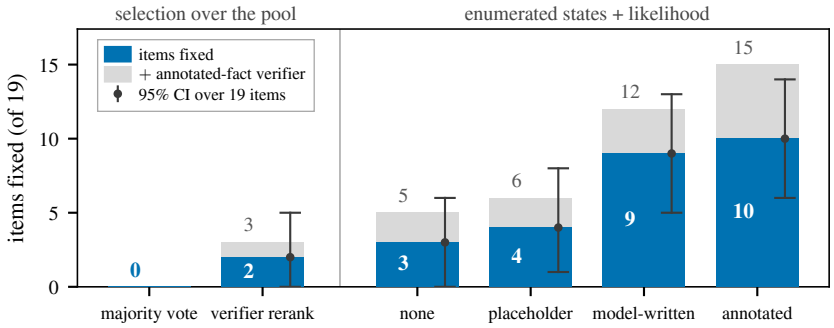


Figure 5: **MuSR `object_placements` on the certified set.** *Right:* the enumerated states take their move sequence from the facts that MuSR's generator provides, so every bar is marked [ORACLE]; the per-step sentence varies from none to annotated.

every character is wrong at 25 forks, and moving it only for the character who moved it is wrong at 23.

The step that sets the answer is the last step at which the annotation changes the queried character's belief about the queried object. The queried character sees no later event involving the object, so the step is decisive regardless of the previous table (Section 3.2). On 11 or 12 of the 19 certified items (Appendix C.2), every sampled chain that reaches this step on the majority path writes the same table, which is wrong in the queried entry. Each of these steps therefore has a consistent error.

With the annotated-fact verifier, state decoding fixes more items than the same verifier over the sampled pool. The enumerated support (S_3, Table 1) is built from the annotated move sequence: at each step with a move, it holds one table for each set of characters who may have seen the move [ORACLE]. The scoring prompt at each step also states the step's move in a per-step sentence of a fixed template. The sentence has four settings (Figure 5): none, a placeholder that keeps the event index and withholds its content, a model-written sentence, and the annotated sentence. The model's own teacher-forced likelihood scores each table given the previous state and the sentence. An *annotated-fact verifier* rejects every state in which the queried character's belief about the queried object is a location that never appears for this belief in the annotated tables [ORACLE]. In the annotated-sentence setting, state decoding with this verifier fixes 15/19 $[11, 18]$, and the same verifier over the sampled pool of 256 chains per item fixes 3/19 $[0, 6]$. The 1024 answers sampled for certification include a correct one on 15 of the 19 items (Appendix C.2). Besides the support, these two runs differ in the search, the per-step sentence and the budget (Appendix A.2). Likelihood scoring alone fixes 3, 4, 9 and 10 of the 19 items in the four settings, and every method over the sampled pool fixes at most 3. A uniform draw over the terminal states of the enumerated support uses no scoring and fixes 8.7 items in expectation. Likelihood scoring is indistinguishable from this draw in the model-written and annotated settings and falls below it in the other two (Appendix C.2).

The likelihood picks a table by the sentence, not by who saw the move. Among the candidate tables at a step with a move, the likelihood mostly picks one of the two that ignore who saw the move: the previous table unchanged, or the table that moves the object in every character's belief. Without a sentence or with the placeholder it picks the first, at 51 and 50 of 57 such steps; with the annotated sentence it picks the second, at 49 of 57. When the annotated successor is neither of these tables, the likelihood picks it at no more than one step in any setting. The items that likelihood scoring fixes therefore depend on whether the queried character saw each move of the queried object. On the same support, a rule that never updates any belief answers 6 items correctly, including every item fixed without a sentence. The rule that moves the object for every character answers 12, including every item fixed with the annotated sentence.

4.4 BELIEF-R: THE OUTPUT DOES NOT SEPARATE THE TWO READINGS

Belief-R (Wilie et al., 2024) adds one premise to a completed inference and asks whether the conclusion still holds (Table 5). The answer depends on how the added premise is read. If it is

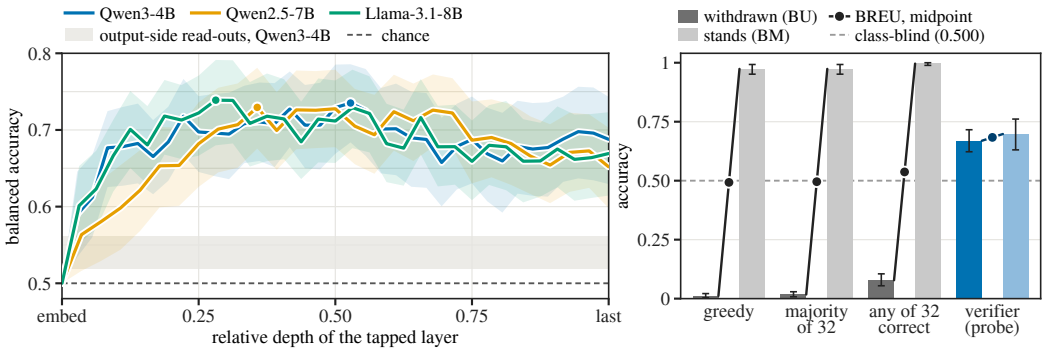


Figure 6: **Belief-R.** *Left:* out-of-fold balanced accuracy of a linear probe; dots mark each curve's peak, and the gray band spans three read-outs of Qwen3-4B's output and an external NLI judge, two of which are AUCs. *Right:* the verifier takes the probe's verdict as the state decision. Bands and whiskers are 95% bootstrap intervals.

a prerequisite that the original inference needed, the conclusion must be withdrawn; if it is an alternative route to the same conclusion, the conclusion stands. The benchmark's metric is BREU, the equal-weight mean of accuracy on items whose conclusion must be withdrawn (BU) and items whose conclusion stands (BM). In the construction of Section 3.4, Belief-R has $T{=}1$: the state at the single step is the reading, and the enumerated support holds both readings on every item. Any choice between the two readings that ignores the item has a BREU of 0.500 (the class-blind floor).

The model applies a given reading, but its likelihood picks one by wording. Greedy decoding and majority voting do not rise above the class-blind floor (Appendix D). Both answer that the conclusion stands on nearly every item, whether or not it must be withdrawn. On 0.078 [0.058, 0.097] of BU items, at least one of the 32 sampled answers takes the reading that withdraws the conclusion; on the others, none does. Since $T{=}1$, the single step is decisive, and each of the other BU items has a consistent error (Section 3.2). The E_2 scoring uses the model's likelihood of the two readings, and its AUC is 0.526 [0.476, 0.575], indistinguishable from chance. The likelihood picks a reading by its wording, not by whether the added premise is a prerequisite. Under the reported wording, the reading in which the conclusion stands has the higher likelihood on every item, and another wording reverses the preference (Appendix D). The model's zero-shot self-report and its in-context choice of reading are also near chance, as is an external NLI judge given the same text (Figure 6, left). When the prompt states the correct reading, the model answers correctly on 0.994 of BU items in modus ponens form, against 0.006 without it. The failure is therefore in choosing the reading, not in applying it. The mid-layer states of three models from two families separate the two readings: a linear probe fitted on human labels reaches balanced accuracy 0.692, 0.716 and 0.734 [ORACLE]. Qwen3-4B's 0.692 is below the pre-registered threshold of 0.70. Sentence length alone also separates the two readings, and a probe's accuracy does not show that the model uses the distinction (Appendix D).

Used as a verifier, the probe gains on one class and loses on the other. When the probe's verdict is the state decision on every item, BREU rises to 0.6835, which exceeds the ceiling of any rule over the sampled pool (any@32) by 0.1465 [0.1043, 0.1878] [ORACLE]. A sentence-length rule fitted on the probe's labels and folds reaches a higher BREU than the verdict (Appendix D). The verdict differs from the pool's answers only where it withdraws the conclusion. Accuracy therefore rises on BU items and falls on BM items (Figure 6, right).

5 Related Work

Selection and search. Self-consistency (Wang et al., 2023), best-of-N (Chen et al., 2021), self-certainty (Kang et al., 2025) and test-time scaling by repeated sampling (Brown et al., 2024; Snell et al., 2025) select among whole chains (S_1 in Table 1). Tree-of-Thoughts (Yao et al., 2023), MCTS-style planners (Hao et al., 2023; Wan et al., 2024), twisted sequential Monte Carlo (Zhao et al., 2024),

Self-Eval beam search (Xie et al., 2023), Q-guided stepwise search (Lin et al., 2025) and A* search with a PRM (Chatziveroglou, 2025) draw candidates from the model at each step (S_2) and score them with a value such as a process reward model (Lightman et al., 2024; Zheng et al., 2025; Zhang et al., 2025). Self-revision (Madaan et al., 2023; Shinn et al., 2023) draws the revised step from the same model. Where majority voting fails, the samples often contain the correct answer (Yin et al., 2024). Yu et al. (2025) call a step a generation failure when no candidate can still reach a correct solution, and a selection failure when one can but is not chosen. They find selection failures more common in mathematical reasoning. A consistent error is a generation failure in which every sampled chain writes the same wrong successor. Tan et al. (2025) find that consistency- and probability-based detectors struggle on its answer-level form. Our analysis holds the model fixed and asks which component the reasoning lacks. On MuSR and Belief-R, once enumeration supplies the missing candidate, the likelihood still does not select it. Appendix E runs three of these decoders on our items.

Explicit state tracking. SymbolicToM (Sclar et al., 2023) and PDDL-Mind (Yi et al., 2026) track beliefs in explicit states and decide by rule who sees each action. The PDDL-Mind authors attribute language models' theory-of-mind failures mainly to unreliable implicit state tracking. Our enumerated support (S_3) also holds explicit states, but the model's likelihood picks among them. The trackers' rules supply what the analysis finds missing on MuSR: the belief update by who saw the move.

Changes to the model. Mozer et al. (2026a) attribute the state-tracking error in the bank example of Section 1 to the feedforward structure of the transformer. Recirculation (Mozer et al., 2026b) mixes a deep layer's activation back into a shallow layer at inference time, and multi-token prediction (Gloeckle et al., 2024) changes the training objective. Masked diffusion models (Nie et al., 2025) can decode in any order; on math and code benchmarks, doing so gives them a lower Pass@k than left-to-right decoding (Ni et al., 2026). We test neither the structural account nor whether these changes reduce consistent errors (Section 6).

6 Discussion

Conclusion. Most failures that more samples do not repair on MuSR and Belief-R are consistent errors: at a decisive step, every sampled chain that reaches it writes the same wrong successor. At that step, selection or search over the pool can pick only a wrong candidate. State decoding shows what each task lacks. On Countdown the resampled support holds a path to the target, but frequency scoring does not pick it. On MuSR and Belief-R the sampled chains lack the correct state, and once enumeration supplies it, the likelihood still does not pick it. With Qwen3-4B, only signals outside the likelihood pick the correct candidate above chance. The search adds no measurable gain on Countdown. Where the answer determines the derivation, the split of Section 3.1 shows whether the pool lacks the correct answer, and so whether the support must change, before any decoder is chosen.

Limitations and future work. Our analysis stops at what the reasoning process makes observable: what the chains write at a decisive step, and which candidate the likelihood picks. Why the model's belief updates do not follow who saw a move, and why its likelihood follows the wording of the candidates rather than their content, remain open. All decoding runs use one model family and one prompt format. The MuSR results cover one subtask and 19 certified items, and the Belief-R results cover one construction of the benchmark (Appendices C.3 and D). These results do not show which component other tasks and models lack, or how often consistent errors occur on them. A decisive step is defined on states, so the framework applies only where the reasoning can be written as a sequence of discrete states. We define the states by hand per task and assume that each holds what the answer needs. Open-ended reasoning would first require extracting states from the chains and finding the decisive steps. SAVI is one instance of the framework. It locates what the reasoning lacks with [ORACLE] components. Its enumerated support on MuSR and Belief-R is built for each item before decoding and holds every alternative of one kind that we choose per task. The argument of Section 3.2 holds for any instance that meets its two conditions. Rather than a better selection rule or search, a decoder that repairs consistent errors needs the two missing components. One is a support that generates candidate states at decisive steps from the chain so far. The other is a scoring that does not come from the likelihood of the states' text. The signals that pick the correct candidate in our experiments come from exact checks, annotations or labels. Learning such a scoring without them is the main open problem. State decoding can test a learned scoring against the [ORACLE] one.

AI USE STATEMENT

Generative AI tools assisted with language polishing, literature search, and data visualization. No generative AI tools were used for synthetic data generation, theoretical models or conceptual frameworks, mathematical claims, hypotheses, experiment design, method implementation, or dataset cleaning and reformatting. This work involves no proofs and no qualitative or thematic data analysis. The authors proposed the research question and the method, determined the conclusions, and checked all AI-assisted work before using it. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Eric J. Bigelow et al. Forking paths in neural text generation. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. arXiv:2412.07961.

Paul C. Bogdan et al. Thought anchors: Which LLM reasoning steps matter?, 2025. arXiv:2506.19143.

Bradley Brown et al. Large language monkeys: Scaling inference compute with repeated sampling, 2024. arXiv:2407.21787.

Giannis Chatziveroglou. A*-decoding: Token-efficient inference scaling, 2025. arXiv:2505.13672.

Fan Chen et al. The coverage principle: How pre-training enables post-training. In *The Fourteenth International Conference on Learning Representations (ICLR)*, pp. 112225–112293, 2026. arXiv:2510.15020.

Mark Chen et al. Evaluating large language models trained on code, 2021. arXiv:2107.03374.

Andrew Estornell and Yang Liu. Multi-LLM debate: Framework, principals, and interventions. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Kanishk Gandhi et al. Stream of search (SoS): Learning to search in language, 2024. arXiv:2404.03683.

Fabian Gloeckle et al. Better & faster large language models via multi-token prediction. In *International Conference on Machine Learning (ICML)*, pp. 15706–15734, 2024.

Shibo Hao et al. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8154–8173. Association for Computational Linguistics, 2023. arXiv:2305.14992.

Ari Holtzman et al. Surface form competition: Why the highest probability answer isn't always right. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7038–7051, 2021.

Chani Jung et al. Perceptions to beliefs: Exploring precursory inferences for theory of mind in large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 19794–19809, 2024.

Zhewei Kang et al. Scalable best-of-N selection for large language models via self-certainty. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, 2025. arXiv:2502.18581.

Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.14901.

Hyunwoo Kim et al. Hypothesis-driven theory-of-mind reasoning for large language models. In *Conference on Language Modeling (COLM)*, 2025. arXiv:2502.11881.

Hunter Lightman et al. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, pp. 39578–39601, 2024.

Zongyu Lin et al. QLASS: Boosting language agent inference via Q-guided stepwise search. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 37942–37958, 2025. arXiv:2502.02584.

Aman Madaan et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pp. 46534–46594, 2023. arXiv:2303.17651.

Michael C. Mozer et al. The topological trouble with transformers, 2026a. URL https://arxiv.org/abs/2604.17121.

Michael C. Mozer et al. Recirculation, 2026b. URL https://arxiv.org/abs/2608.17981.

Zanlin Ni et al. The flexibility trap: Rethinking the value of arbitrary order in diffusion language models. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. arXiv:2601.15165.

Shen Nie et al. Large language diffusion models. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, pp. 56354–56392, 2025. doi: 10.52202/085713-1689. arXiv:2502.09992.

Hadas Orgad et al. LLMs know more than they show: On the intrinsic representation of LLM hallucinations. In *The Thirteenth International Conference on Learning Representations (ICLR)*, pp. 66880–66913, 2025. arXiv:2410.02707.

Nikhil Prakash et al. Language models use lookbacks to track beliefs. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. arXiv:2505.14685.

Melanie Sclar et al. Minding language models' (lack of) theory of mind: A plug-and-play multi-character belief tracker. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13960–13980. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.780.

Noah Shinn et al. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pp. 8634–8652, 2023. arXiv:2303.11366.

Charlie Snell et al. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations (ICLR)*, pp. 10131–10165, 2025. arXiv:2408.03314.

Zayne Sprague et al. MuSR: Testing the limits of chain-of-thought with multistep soft reasoning. In *International Conference on Learning Representations (ICLR)*, pp. 14670–14728, 2024.

Felix Stahlberg and Bill Byrne. On NMT search errors and model errors: Cat got your tongue? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3356–3362, 2019.

Hexiang Tan et al. Too consistent to detect: A study of self-consistent errors in LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 4755–4765, 2025. arXiv:2505.17656.

A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967. doi: 10.1109/TIT.1967.1054010.

Ziyu Wan et al. AlphaZero-like tree-search can guide large language model decoding and training. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 49890–49920. PMLR, 2024. arXiv:2309.17179.

Ante Wang et al. Don't get lost in the trees: Streamlining LLM reasoning by overcoming tree search exploration pitfalls. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23946–23959, 2025.

Xuezhi Wang et al. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pp. 24824–24837, 2022. arXiv:2201.11903.

Bryan Wilie et al. Belief revision: The adaptability of large language models reasoning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 10480–10496, 2024.

Yuxi Xie et al. Self-evaluation guided beam search for reasoning. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. arXiv:2305.00633.

Shunyu Yao et al. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, pp. 11809–11822, 2023. arXiv:2305.10601.

Qiutong Tony Yi et al. PDDL-Mind: Large language models are capable on belief reasoning with reliable state tracking, 2026.

Zhangyue Yin et al. Aggregation of reasoning: A hierarchical framework for enhancing answer selection in large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 609–625, 2024. arXiv:2405.12939.

Fei Yu et al. Scaling flaws of verifier-guided search in mathematical reasoning, 2025. arXiv:2502.00271.

Zhenru Zhang et al. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.547. arXiv:2501.07301.

Stephen Zhao et al. Probabilistic inference in language models via twisted sequential Monte Carlo. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 60704–60748. PMLR, 2024. arXiv:2404.17546.

Chujie Zheng et al. ProcessBench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1009–1024. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.50. arXiv:2412.06559.

Wentao Zhu et al. Language models represent beliefs of self and others. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62638–62681. PMLR, 2024.

A SEARCH AND BUDGET

A.1 THE SEARCH AND THE CANONICALIZER Φ

The canonicalizer Φ maps a derivation to a key, and the trellis treats two derivations with the same key as one node. On *Countdown* the key is the sorted multiset of remaining values together with the target. Two derivations that share a key leave an identical subproblem, so Φ cannot fuse two states that differ; at worst it leaves equivalent states unmerged. Wang et al. (2025) merge states in tree search by clustering their text embeddings, so their merging is approximate. On Game of 24, Tree-of-Thoughts gives the model the remaining numbers at each node, and each node is the input together with the thoughts so far (Yao et al., 2023).

Keeping one survivor per node also treats two histories that reach the same state as interchangeable. We call this the same-render condition: the model's later steps depend on the state's text alone. It holds by construction, because the model samples or scores each step given only the task input, the state that the step extends and, on MuSR, the step's sentence. Under this condition, and with a beam K at least as wide as each layer, the recursion of Section 3.4 is exact. Section 3.2 uses this exactness to count every remaining error as a support or a scoring error. Stahlberg & Byrne (2019) use exact search in the same way to separate search errors from model errors in machine translation. Whether dropping the history costs accuracy is not tested.

A.2 BUDGET ACCOUNTING

On Countdown, decoders are compared at matched token counts, never at matched candidate counts: a selection baseline is given the same token budget as the decoder it is compared against, and the token count is the whitespace-split length of the rendered text. A best-of-N chain of m steps counts $2m{+}1$ words, while each single-step candidate of state decoding and of masked best-of-N counts 3. Five steps therefore count 11 words under best-of-N and 15 under state decoding, so this accounting favors the baseline. The baseline is a *verified* best-of-N rather than an unverified draw, so a gain cannot come from spending more samples. The MuSR runs of Section 4.3 are not matched in tokens; Table 2 lists their token counts. With a separately trained feasibility classifier in place of the exact verifier, the matched-token margin at $k{=}6$ is -0.583 $[-0.83, -0.33]$ when the classifier's score is soft-weighted, and no item is solved when the score is thresholded into a mask.

Compute counts. Compute is reported as counts that the configuration fixes, as the minimum, median and maximum over items. On MuSR (Table 2, top), a call is one sampled sequence for majority voting and one forward pass for enumerated-state decoding. Enumeration scores each edge of the trellis once, so its number of calls is fixed by the number of states in each layer, which the table lists from the initial state to the last step. Its prefill tokens are the prompt and candidate tokens that each forward pass reads, a different quantity from the generated tokens of voting. On Countdown (Table 2, bottom), a call draws one whole chain for greedy decoding, self-consistency and the token-matched draw of Section 4.1, and one step of 8 sequences for state decoding. Calls therefore count different things across decoders, and self-consistency is counted as 8 calls. Countdown tokens are whitespace-split words, as in the comparisons above, and are not comparable to MuSR tokens. The token-matched draw is given the token count of state decoding on the same item, so their token columns agree by construction.

Wall-clock ratios. Absolute times depend on batching and on other jobs on the shared GPUs. Table 3 therefore reports only ratios between two decoders run on the same item, on one GPU and in one period. On MuSR, majority voting ($N{=}64$, batch 16) and enumeration in the annotated-sentence setting (batch 8) alternate item by item on one loaded model, over the 19 certified items and the three knowledge controls (Appendix C.1). On Countdown, the four decoders of Table 2 run back to back on each of its items. State decoding generates the 8 candidate successors of a node in one batch, while the other decoders generate one step of one chain at a time. This difference sets most of the ratio between state decoding and the token-matched draw. The time of state decoding also includes the exact verifier, a search whose memo table fills over the items of each depth. Later items of a depth therefore run faster, by an amount we did not measure. Of the ten intervals, only the one for greedy decoding at $k{=}4$ contains 1.

MuSR decoder	states per layer	calls per item	items	tokens per item min	tokens per item median	tokens per item max
majority voting	—	64	19	14,737	22,522	34,129
enumeration	$1, 8, 8, 64$	100	1	98,045	98,045	98,045
	$1, 8, 27, 216$	289	1	402,764	402,764	402,764
	$1, 8, 64, 64$	289	9	302,703	380,563	435,586
	$1, 8, 64, 216$	585	8	667,281	728,750.5	879,556

k	Countdown decoder	calls per item min	calls per item median	calls per item max	tokens per item min	tokens per item median	tokens per item max
4	greedy	1	1	1	7	7	7
4	self-consistency	8	8	8	52	56	56
4	state decoding, K=8	1	4	13	24	96	312
4	token-matched draw	4	14.5	45	28	101.5	315
5	greedy	1	1	1	9	9	9
5	self-consistency	8	8	8	70	72	72
5	state decoding, K=8	1	9	20	24	216	480
5	token-matched draw	3	24.5	54	27	217.5	482
6	greedy	1	1	1	11	11	11
6	self-consistency	8	8	8	80	88	88
6	state decoding, K=8	18	26.5	33	432	636.5	792
6	token-matched draw	40	59	74	440	641	798

Table 2: **Compute counts per item**, as minimum, median and maximum. *Top:* MuSR, the 19 certified items. *Bottom:* Countdown, 12 items per depth k.

	majority voting / enumeration		
MuSR	$2.707\ [2.224, 3.047]$		
	state decoding / other decoder		
Countdown	greedy	self-consistency	token-matched draw
k=4	$1.451\ [0.452, 2.879]$	$0.221\ [0.056, 0.360]$	$0.120\ [0.110, 0.125]$
k=5	$2.978\ [1.819, 5.396]$	$0.369\ [0.224, 0.670]$	$0.121\ [0.118, 0.123]$
k=6	$6.993\ [6.202, 7.484]$	$0.874\ [0.792, 0.906]$	$0.118\ [0.116, 0.120]$

Table 3: **Wall-clock ratios per item**, as the median over items with a paired bootstrap 95% interval. *Top:* MuSR, 22 items. *Bottom:* Countdown, the items of Table 2.

B COUNTDOWN

Stream of Search (Gandhi et al., 2024) also uses Countdown. It trains language models on the traces of symbolic search.

B.1 THE GENERATOR'S ADAPTER AND THE LEARNED SIGNALS

The LoRA adapter. We call the model that proposes moves on Countdown (Section 4.2) the generator. It is Qwen3-4B with thinking disabled and a LoRA adapter of rank 16, α=32 and dropout 0.05. The adapter is attached to the query, key, value and output projections and to the gate, up and down projections. The base weights stay frozen. Each training example is the decoding prompt filled with one state, followed by one move in the output format of that prompt and the end-of-sequence token. The loss is cross-entropy on the move and end-of-sequence tokens, plus 0.05 times $\mathrm{KL}(p_{\text{base}} \,\|\, p_{\text{adapter}})$ between next-token distributions on four fixed English sentences. Training runs for 6 epochs with AdamW, learning rate 10^{-4}, batch size 8, no learning-rate schedule and training seed 0. The examples come from 207 solvable items with four numbers, generated with item seed 0. For each item, an exact solver finds one solution path. The path has three states before the target

is reached, and each of them contributes examples. At each of these states, the exact reachability check keeps the legal moves after which the target is still reachable. Up to 4 of these moves are drawn without replacement, and each gives one example. The training set thus has 1036 examples. The exact check is used here to build training data and not during decoding, so the ORACLE mark of Section 4 does not cover this use. The test items have six numbers, so no test item is a training item. Some training states can also be reached from test items. We did not check whether any of them occurs when decoding the test items.

The learned head and the cheap signal. Both signals are scikit-learn multilayer perceptrons. Each estimates the probability that the target is reachable from a state. Their inputs are standardized, and training stops early on a validation split. The learned head has hidden layers of 256, 128 and 64 units and trains for at most 500 iterations. The cheap signal has hidden layers of 64 and 32 units and trains for at most 300 iterations. Other settings are the library defaults. The learned head reads 40 features of the state, and the cheap signal reads the first 22 of them. These 22 are statistics of the remaining numbers and the target, followed by the numbers themselves, sorted and padded to six. The statistics include the count, sum, extremes and spread of the numbers, the difference between their sum and the target, and the counts of numbers that divide, equal or exceed the target. The other 18 features add statistics of the same kind and counts of pair results that equal or divide the target. A pair result combines two of the numbers by one operation. No feature uses more than one operation. The label of a state is whether the target is reachable from it. The exact check computes these labels. It is not used when decoding with either signal, so the ORACLE mark does not cover this use either. The training items are the first eight solvable items with six numbers from each item seed 1000 to 1007. The test items come from item seeds 0 to 5. Nineteen of the 64 training items are held out from fitting. On each of the remaining 45 training items, states are collected by an expansion that uses the generator's proposals and no verifier. The expansion runs for at most 5 layers. At each layer, it samples 8 proposals at temperature 1 from each state in the frontier and keeps the 8 most frequent distinct successors across the frontier. Each distinct state visited on an item, including the initial state, is one training example. Both signals are fitted on the same 1778 states. In decoding, the value V of a successor enters Equation (1) as $\lambda \log \max(V, 10^{-3})$, and no successor is pruned. The values of λ are those of Figure 4(b). No training item shares its identifier with a test item, and none has the same numbers and target as a test item. Some test items share their target with a training item, so a state can occur on both. We did not check whether any training state occurs when decoding the test items.

B.2 PATHS OUTSIDE THE SAMPLED POOL

The 45 items of Section 4.2 are those on which none of 64 sampled chains is correct. Under another sampling seed, 64 chains solve 17 of them, and 1024 chains solve all of them. On these items, whether state decoding uses frequency alone or adds a verifier, every path it solves contains a step that none of the original 64 chains took. We call a path stitched if each of its steps was taken by some sampled chain but no single chain took them all. Across the six generators we tried, at most 1 of the 45 items is solved by a stitched path. Each generator's per-step distribution is too peaked to cover the deep steps that a stitched path would join. Fine-tuning the generator on several correct derivations per item raises the coverage of depth-1 steps from 0.05 to 0.33 and of depth-2 steps from 0.008 to 0.035, and still no item is solved by a stitched path ($0/45$). A correct depth-5 path needs all five of its steps covered, and each deep step is covered about 0.03 of the time, so the chance that all five are covered is negligible. What blocks stitching on Countdown is this joint coverage, not the coverage of single steps.

Uniform proposals. When uniform draws from the legal moves replace the model's proposals, state decoding solves at most 2 of the 45 items with frequency alone and 39 to 42 with the exact verifier, over five seeds. Most solutions under the exact verifier therefore do not depend on the model's proposals. In these runs, each node draws 8 successors with replacement, and moves that give the same result count as one. The beam, the merging and the scoring are those of the runs with the model's proposals. When states tie for the last places in a layer, the runs keep the first in canonical order; with random tie-breaking instead, frequency alone still solves at most 2 items. The runs with the model's proposals use one decoding seed. The 45 items were selected because the model's 64 chains fail on them, so these counts do not describe all $k{=}6$ items.

On a synthetic lattice, decoding saves draws. On this synthetic task a derivation is a path through an additive lattice, and the generator takes a correct step with probability p. We compare decoding that uses a learned value as a soft weight with selection over $M{=}256$ chains. The correct path that the decoder returns has probability between $10^{-11.5}$ and 10^{-10} under the generator, so sampling would need on the order of 10^{11} draws to produce it. Every correct path that decoding returns is new, and 23% to 36% of them are stitched ($p{=}0.5$ and 0.6, with merging). The learned value produces these paths: at $p{=}0.5$ the decoder recovers 0.23 of the items that selection misses, against 0.01 with frequency alone. Selection solves the same items once it draws about $10^{3.4}$ to $10^{4.2}$ chains. An additive lattice has many correct paths, so a rare path does not imply a rare answer. On this task decoding reduces the number of draws and does not reach answers that sampling cannot.

B.3 The learned head's errors

The gap between the learned head and the exact verifier on Countdown is not explained by the head's false-negative rate, and a larger training pool does not close it. With the exact verifier corrupted at random to a false-negative rate of 0.3, state decoding still solves 0.911 of the 45 items of Section 4.2. The learned head has a false-negative rate of 0.289 on held-out deep states, lower than that of the corrupted verifier, and solves 0.400. As in Section 4.2, an item counts as solved by the head if either of the two weights λ solves it. Training the head on $16\times$ as many labeled examples raises its solved fraction by $+0.156$ $[+0.067, +0.252]$, pooled over three training seeds. At the larger pool, the lower bound of the solved fraction stays below the pre-registered threshold of 0.70.

Correcting the learned head at the boundary. We rerun state decoding with the learned head on the 45 items of Section 4.2 and replace its judgment with the exact verifier's on one class of judgments at a time [ORACLE]. A boundary judgment scores a successor of a reachable state whose resampled successors include both reachable and unreachable ones, so the step from that state is decisive (Section 3.2). Boundary judgments are 0.38 of the head's 10795 judgments. On them the head rejects 803 of 1544 reachable successors and accepts 594 of 2559 unreachable ones; on all other judgments it rejects 10 of 517 and accepts 122 of 6175. The head's errors are therefore concentrated at decisive steps. The runs in this paragraph count solved items in the same way. State decoding solves 0.400 with the head alone and 0.978 with the exact verifier. With the exact verifier on boundary judgments only it solves 0.978, and on all other judgments only 0.778. The paired differences from the head alone are $+0.578$ $[+0.422, +0.711]$ and $+0.378$ $[+0.244, +0.533]$, and boundary-only exceeds other-only by $+0.200$ $[+0.089, +0.311]$. Correcting the head at boundary judgments alone thus recovers the exact verifier's result. Correcting only the other judgments also raises the solved fraction above that of the head alone, so the head's errors off the boundary also cost accuracy. At every parent state that two or more of these runs expand, the runs expand the same candidates.

C MuSR

Table 4 shows the candidate tables at one step of a certified item and the prompt used to score them.

C.1 The certified set, and how common its failure is

We construct the certified set through the funnel of Figure 7. F1 and F2 keep 26 items, and F3 and F4 leave the 19 certified items. The 26 items that pass F2 are hard: median per-item sampling correctness is 0.071 at $N{=}1024$, and 4 of the 26 have no correct answer among the samples. The self-extraction control (F3) removes 5 easier items, with median correctness 0.316 against 0.044 on the remaining items. We did not score the self-extracted facts against the annotation, so F3 does not show which extracted facts are wrong. Three further items stay wrong when the annotated facts are given. They serve as knowledge controls (Appendix C.2).

How common is the failure this set isolates? The certified set is selected for voting failure, so on its own it does not show how often the failure occurs. We therefore count the failure on the unfiltered benchmark slice. Call a step *unanimous-wrong* if every sampled chain writes the same, incorrect successor state. A step is *judgeable* if at least 64 sampled chains reach the state before it, so that the rule of three bounds the mass of any unseen successor at $3/64$; an item is judgeable

	belief about the map			
saw the move	Adam	Rachel	Teddy	
none	backpack	backpack	backpack	previous table unchanged
Adam	picnic table	backpack	backpack	
Rachel	backpack	picnic table	backpack	
Teddy	backpack	backpack	picnic table	mover only
Adam, Rachel	picnic table	picnic table	backpack	
Adam, Teddy	picnic table	backpack	picnic table	annotated
Rachel, Teddy	backpack	picnic table	picnic table	
all three	picnic table	picnic table	picnic table	moved for every character

```
[story, 686 words]

Which location is the most likely place Adam would look to find the map given the story?

Pick one option:
1. backpack
2. picnic table
3. tent

Track what every character believes after each event. The belief state after event 0 is:
BELIEF[0]: {"Adam": {"map": "backpack", "sunscreen": "tent"},
    "Rachel": {"map": "backpack", "sunscreen": "tent"},
    "Teddy": {"map": "backpack", "sunscreen": "tent"}}

Event 1: the map is moved to the picnic table.

Write exactly one line recording what every character now believes about where every
    tracked object is, in the same format.
BELIEF[1]: {"Adam": {"map": "picnic table", "sunscreen": "tent"},
    "Rachel": {"map": "backpack", "sunscreen": "tent"},
    "Teddy": {"map": "picnic table", "sunscreen": "tent"}}
```

Table 4: **MuSR: the enumerated candidates at one step.** Certified item 0046-q0 at step 1, where Teddy moves the map to the picnic table. The question asks where Adam would look for the map, and the correct option is the picnic table. No later annotated table changes Adam's belief about the map, so this is the step that sets the answer (Section 4.3). *Top:* one candidate successor of the initial table for each set of characters who may have seen the move. In every candidate, all three characters believe that the sunscreen is in the tent. *Bottom:* the prompt in the annotated-sentence setting, followed by the line of the annotated candidate. Likelihood scoring appends the line of each candidate to this prompt and scores only that line. Long lines are broken for display.

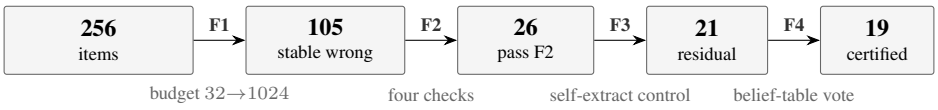


Figure 7: **The certification funnel.** F1 keeps the items whose majority answer stays wrong at a $32\times$ larger sample budget (107 wrong at $N{=}32$, 105 at $N{=}1024$). F2 keeps the items that pass four checks: (i) the wrong majority is statistically stable; (ii) the annotated facts turn the answer right; (iii) no model on a Qwen2.5 ladder from 0.5B to 7B solves the item; (iv) a second seed reproduces the failure. F3 removes the 5 items fixed once the model extracts the facts for itself. F4 re-runs majority voting with chains that write belief tables: the 19 of the 21 that remain voting-wrong form the certified set.

if it has a judgeable step. On the unfiltered slice, 32/40=0.800 judgeable items (story bootstrap $[0.599, 0.975]$) contain at least one unanimous-wrong step, and 66/115=0.574 of judgeable steps are unanimous-wrong. Only the 40 items sampled with 256 chains are judgeable; in the slices with 24 chains per item no step reaches 64 chains. Of the 66 steps, 43 copy the previous table and 23 record a different set of entries from the annotated table; at none do the chains agree on a different wrong update. Two artifacts can enter these counts. A table that names an object differently from the annotation counts as wrong, and a copied step may be one whose move the chains write in a later line (Appendix C.2). Containing a unanimous-wrong step is not the same as answering wrong, and

voting answers some of these items correctly. Steps at which every chain copies the previous table, or records a different set of entries, are therefore common on the unfiltered slice. This count does not check whether a step is decisive, so it does not count consistent errors (Section 3.2).

C.2 Further interventions and mechanism checks

The step that sets the answer. Section 4.3 takes the step that sets the answer to be the last step at which the annotation changes the queried character's belief about the queried object. Five of the 19 certified items have no such step, and on the other 14 the step is judgeable (Appendix C.1). On 12 of the 14, every chain that reaches the step on the majority path writes the same table, and that table is wrong in the queried entry. It copies the previous table on 7 items and records a different set of entries on 5; on none do the chains agree on a different wrong update. On the other 2, the chains disagree. On item 0042-q0 the step changes the belief between two malformed location strings of the dataset, which may name the same place; if they do, that item has no such step, and the count is 11. On all 14 items the previous table on the majority path already differs from the annotated one, so these counts do not show an error made from a correct history. The previous table does not affect whether the step is decisive. On every item except 0043-q2, the queried character moves the queried object at this step or sees another character move it. On 0043-q2, the character instead sees the object while moving another one. In the annotation, the queried character sees none of the later events that involve the queried object. Forking tokens (Bigelow et al., 2025) and thought anchors (Bogdan et al., 2025) are found by resampling the chain from a position and measuring how the distribution of final answers changes. The definition in Section 3.2 assumes correct later steps, so it does not depend on what the chains write after this step.

Copied steps and later writes. A chain writes the move of step t if, in its table for step t or in any later one, the mover's belief about the moved object equals the new location. A copied step is left out when a write cannot be attributed to its move: when the previous table already holds the new location for the mover, or when the annotation later moves the same object to the same place. This leaves 23 steps on the certified items at which every chain copies the previous table. At 16 of them no chain writes the move later, so at these steps the copy is not a delayed write. The exact match misses a chain that words the location differently, so these 16 steps are an upper bound. Even when any later change of the mover's entry counts as a write, a majority of the chains write the move at only 5 of the 23.

Six unanimous-wrong steps. We examine six unanimous-wrong steps more closely. An earlier analysis chose them, one on each of six items that enumerated-state decoding fixes, among the steps at which the decoded path differs from the path decoded without a per-step sentence. They were not chosen as decisive steps, and five of them are the step that sets the answer. At these steps the rule of three bounds all successor mass other than the shared wrong one at 0.012–0.027. Both the annotated table and the shared wrong successor are scored by teacher-forced likelihood. The condition is the prompt that the chains are sampled from, followed by the tables of the majority path at the earlier steps. These tables differ from the annotated ones. The annotated table carries probability in $[2.4e-38, 3.9e-31]$ against 1e-7–3e-6 for the shared wrong successor. Each quoted probability is for the single canonical line that realizes a state, so summing over paraphrases can only raise a state's total mass and the quoted values are lower bounds on state reachability.

Reweighting the samples does not reach the correct successor either. Sharpening, that is, power sampling with $\alpha{>}1$ (Karan & Du, 2026), moves more mass onto the wrong successor. Flattening ($\alpha{<}1$) would need at least 4.4e5 samples to reach the correct one at the most favorable fork ($\alpha{=}0.25$; 2e11 at $\alpha{=}0.5$), against a feasible cap of 4096, which is 16 times the 256 chains we draw. Paraphrase mass on the correct successor could lower these counts.

Asking the model to revise leaves the six steps unchanged. Two greedy revision rounds at a 2048-token budget change the successor at none of them (0/6) and fix 5/19 items overall, a count inside the pre-registered range from which we draw no conclusion.

Forks and the two model-free rules. The 30 forks of Section 4.3 come from 10 stories. The model's most frequent table is right at 6 forks, against 5 for the rule that moves the object for every character and 7 for the rule that moves it only for the mover. The paired differences are $+1$ $[-2, +4]$

and -1 $[-6, +4]$. The second rule takes the mover from the annotation, which the model must find in the story. Neither rule is the strongest model-free rule.

Further readings behind Section 4.3. Reranking the pool with a learned verifier fixes 2 of the 15 certified items whose 1024 samples include a correct answer. With model-written sentences, the likelihood updates at 20 of the 23 steps whose sentence names the step's move and at 1 of the 23 whose sentence does not, the same dependence on the sentence as in the other settings. Likelihood scoring fixes items in the queried entry only: no path that it decodes matches the annotated tables at every step.

Knowledge controls, and what the written tables measure. Likelihood scoring over the enumerated support fixes none of the knowledge controls (Appendix C.1). One of the three cannot be read out in any state space, so the count is 0/2 in every sentence setting. A written table does not show the model's conditional distribution. At most forks the fully informed model still samples a wrong table, and the order in which the likelihood ranks candidate successors agrees only weakly with how often the model samples them (median Spearman ρ=0.400).

Paired differences behind Figure 5. Each difference is summed over the same 19 certified items, resampled together (B=10000, percentile 95%). The uniform draw of Section 4.3 has per-item expectation $1/2$ on 14 items and $1/3$ on 5.

difference	net	95% interval
placeholder − none	$+1$	$[0, +3]$
model-written − none	$+6$	$[+1, +11]$
annotated − none	$+7$	$[+1, +12]$
model-written − placeholder	$+5$	$[0, +10]$
annotated − placeholder	$+6$	$[0, +11]$
annotated − model-written	$+1$	$[-4, +6]$
none − uniform draw	-5.67	$[-8.33, -2.50]$
placeholder − uniform draw	-4.67	$[-7.67, -1.00]$
model-written − uniform draw	$+0.33$	$[-3.50, +4.17]$
annotated − uniform draw	$+1.33$	$[-2.83, +5.33]$

Other work on belief updates. On ToMi and FANToM, Jung et al. (2024) find that models infer who perceives an event well and infer beliefs from those perceptions poorly. On MuSR, our analysis places the failure in extracting from the story who moved each object and who saw each move (Section 4.3). Prakash et al. (2026) find the mechanism by which a model updates a character's belief when the text states who can see whom. Thought-tracing (Kim et al., 2025) weights belief hypotheses by likelihoods that the model states as one of six verbal levels. The scoring in our analysis reads the teacher-forced likelihood of each enumerated table instead.

C.3 Scope, and the two other domains

The benchmark is public, so contamination cannot be excluded. The fork check of Section 4.3 starts each fork from the annotated table, so it does not measure error accumulation. Enumerated-state decoding scores each edge from the previous table and the step's sentence, while the voting chains condition on the full history. The per-step sentence of Section 4.3 follows a fixed template, for example "*Event t: the keys are moved to the box.*" Certification and decoding were run on `object_placements` only, and the method is not claimed on the two other MuSR domains. In those domains we measured how often the model's likelihood ranks the annotated statement first among the candidates. At the 56 forks of `murder_mystery` where the model writes a wrong statement, the rate is 16/56=0.286, not above the chance rate of $1/2$ (one-sided binomial p=0.9997). For `team_allocation` it is above chance over the capability cells (84/180=0.467 against $1/3$, p=$1.4e-4$) but not at the step that decides the answer (9/20, p=0.19). The likelihood fails the same way on `object_placements`: at the six steps examined in Appendix C.2, it ranks the wrong successor that every chain writes above the annotated one.

```
If John waters the plants, then Jessica will prune the leaves
John waters the plants
If the roses reach full bloom, then Jessica will prune the leaves

What necessarily had to follow assuming that the above premises were true?
(a) Jessica will prune the leaves
(b) Jessica will not prune the leaves
(c) Jessica may or may not prune the leaves

Begin! Reminder to write your final answer as "Final Answer [X]." and fill [X] with
    either a, b, or c.
```

		reply for the reading under which the conclusion	
wording	question after the three premises	is withdrawn	stands
sufficiency	Considering all three statements, is the condition in the first statement enough on its own to bring about the outcome?	No, it is not enough on its own.	Yes, it is enough on its own.
relation	How does the third statement relate to the first?	It states a further condition that is also needed for the outcome.	It states a separate route that on its own leads to the outcome.
certainty	Given all three statements, is what follows certain?	No, what follows is uncertain.	Yes, what follows is certain.

Table 5: **Belief-R: one item and the two prompts.** *Top:* the prompt for a direct answer on the BU item `260-strong`, in modus ponens form. The correct answer is (c). The greedy answer and the 32 sampled answers are drawn with this prompt. Item `259-weak`, whose correct answer is (a), is a BM item with the same first two premises and the added premise "If it rains tomorrow then Jessica will prune the leaves". *Bottom:* likelihood scoring (E_2) appends one question to the three premises and scores the two replies. Both prompts are wrapped in the model's chat template with thinking disabled.

D BELIEF-R

Table 5 shows one item with the prompt for a direct answer and the prompt for likelihood scoring.

The benchmark has a construction artifact: a rule reading only the character-count difference between the added premise and the original ones reaches AUC 0.800. Every probe number is therefore also read on a length-matched subset, where the three models' across-split means fall between 0.60 and 0.64 with lower bounds above 0.5. That subset's pass criterion was relaxed after the results were seen, which departs from the pre-registration. Under the frozen criterion the main model's 0.692 falls 0.008 short of the pre-registered threshold of 0.70, and the across-split means that exceed it come from an exploratory re-split.

The verdict numbers are measured with one model (Qwen3-4B), one prompt and one level of annotator agreement (lv$=5$, the `agreement_lv` column of the released dataset). The probe is fitted and read out of fold on the pre-registered split (lv$=5$ ponens, $n=391$, 152 groups). The three models are probed at 37, 29 and 33 layer positions, so Figure 6 plots relative depth. Three read-outs come from the output of Qwen3-4B: zero-shot self-report, likelihood scoring of the two enumerated readings, and in-context choice. The fourth is an external NLI judge that runs without Qwen3-4B. Likelihood scoring was run under three wordings of the two readings: sufficiency (the reported wording), relation and certainty. The likelihood is not corrected for surface form competition, in which different strings that express the same answer compete for probability mass (Holtzman et al., 2021). With log-probabilities summed over tokens on all 1744 items, the likelihood prefers the reading in which the conclusion stands on every item under sufficiency and certainty, and prefers withdrawal on 0.974 of items under relation. With log-probabilities averaged over tokens on the 782 items at lv$=5$, the AUCs are 0.526 $[0.476, 0.575]$ for sufficiency, 0.380 $[0.330, 0.433]$ for relation and 0.460 $[0.411, 0.509]$ for certainty. None of the three intervals lies above 0.5. Under relation the interval lies below 0.5, so the likelihood ranks the two readings in reverse. The direction differs across models: at $\lambda=0$ under relation, Qwen2.5-7B has a pooled AUC of 0.632 $[0.561, 0.696]$ (Figure 9).

Linear probes on a model's activations also decode the beliefs of story characters (Zhu et al., 2024). Orgad et al. (2025) use a probe to choose among resampled answers, which requires the correct

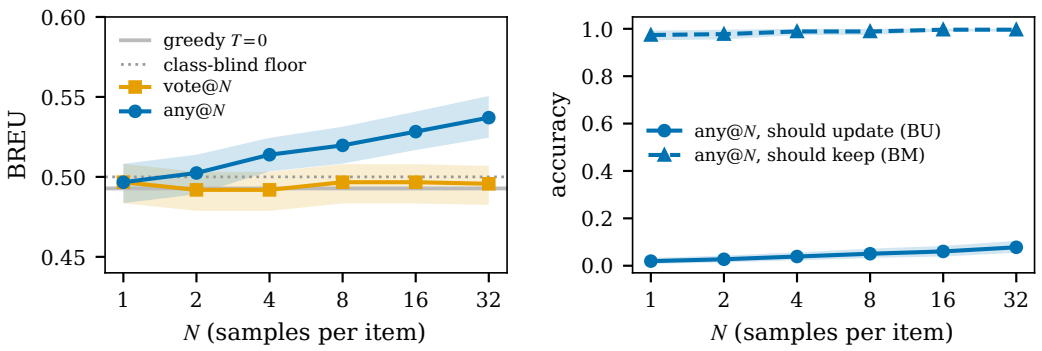


Figure 8: **Belief-R: any@N on nested prefixes of the $N{=}32$ pool.** *Left:* BREU for greedy, vote@N and any@N, with 95% scenario-clustered bootstrap bands; the dotted line is the class-blind floor. *Right:* any@N on the two item classes. The rise is on items that should update; items that should keep are flat from $N{=}16$.

answer to appear among them. The probe here decides between the two enumerated readings, so it does not need the correct reading in the pool. Belief-R has one step, so a consistent error on it is the answer-level self-consistent error of Tan et al. (2025): every sample gives the same wrong answer. They report that probes on the model's own hidden states detect such errors poorly. The probe here separates two readings rather than detecting an error in a sampled answer. In debate among several models, the answers converge to the majority opinion, and that opinion can come from a misconception that the models share (Estornell & Liu, 2024).

The verdict is used as the state decision on all 782 rows of the split, ponens and tollens; the balanced accuracy in Section 4.4 is the ponens half. Its paired intervals over any@32 are $+0.5914$ $[+0.5389, +0.6440]$ on items that should update and -0.2985 $[-0.3619, -0.2351]$ on items that should keep. When the same verdicts are written into the prompt and the model regenerates its answer, BREU is 0.6557. A logistic rule on the character-count difference alone, fitted on the same labels and folds as the probe, scores 0.7462 on BREU over the same rows, above the verdicts' 0.6835 (paired difference $[-0.1181, -0.0089]$). On the length-matched subset, the rule scores 0.5169 $[0.4494, 0.5899]$ and the verdicts score 0.5955 $[0.5337, 0.6545]$. The paired difference on that subset has an interval that contains zero.

The Belief-R pool is $N{=}32$ and is not extended to the $N{=}64$ of the other two sampled tasks. On BREU, greedy decoding scores 0.4928 $[0.4796, 0.5031]$ and majority voting over this pool scores 0.4957 $[0.4825, 0.5069]$. Both intervals contain the class-blind floor of 0.500. Figure 8 shows any@N on nested prefixes of that pool: on BREU it rises by $+0.006$ to $+0.011$ per doubling of N, close to linear in $\log N$, and the rise comes from items that should update, on which any@32 is 0.078. At the observed rate any@64 would be near 0.546, the order of ϵ_{miss} across the three tasks in Figure 3 would not change, and the $+0.1465$ margin of the probe verdict over any@32 (Section 4.4) would need more than twelve doublings of N to close, even at the fastest observed rate.

E Other decoders on the same items

We ran three published decoders on the test items of Section 4 with Qwen3-4B, without thinking. The Countdown and MuSR items were selected because sampling fails on them. Each differs from SAVI in more than one component (Table 1), so each comparison below is a package, and a difference cannot be assigned to one component. Intervals resample items on Countdown, stories on MuSR and scenarios on Belief-R. An interval that contains zero is read as indistinguishable, not as equal.

Self-certainty over the sampled pool. Self-certainty (Kang et al., 2025) scores a sampled answer by the mean divergence of the model's next-token distributions from the uniform distribution. A Borda vote with exponent $p{=}1.2$, fixed before the runs, weights each answer by its rank under this score. The support stays the sampled pool (S_1), and only the scoring changes. The Borda vote can therefore differ from majority voting in correctness only on items whose pool holds both the correct

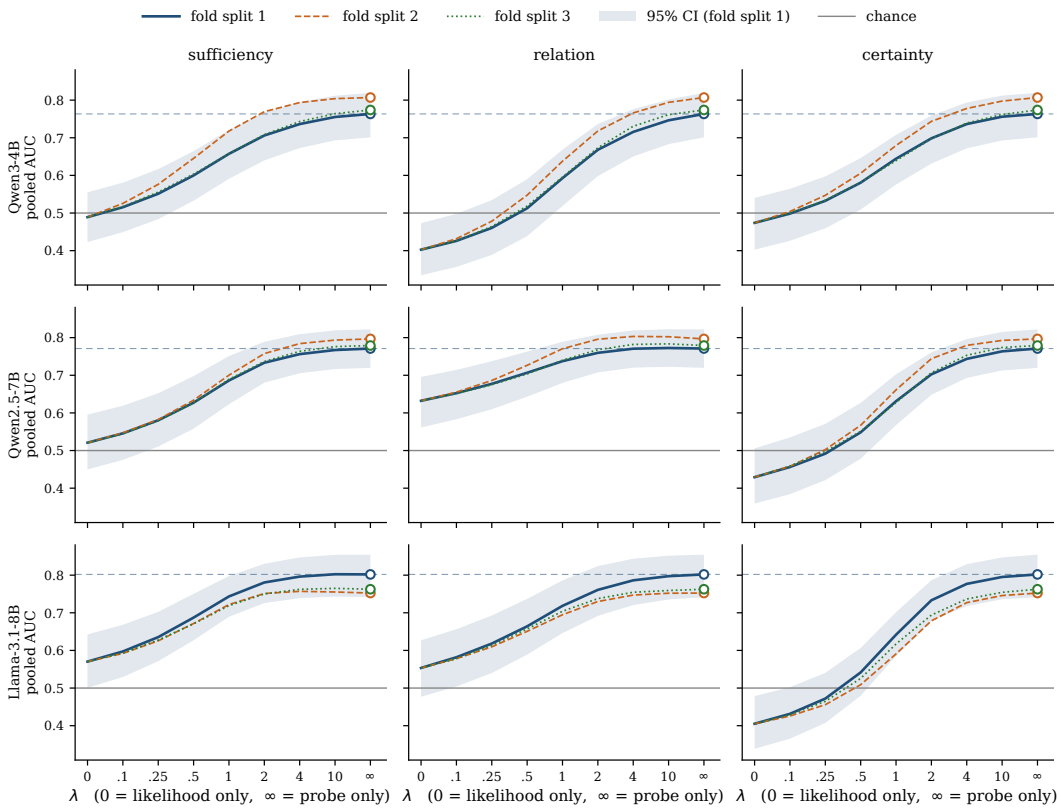


Figure 9: **Belief-R: the likelihood mixed into the probe score.** Each item is scored by $s(\lambda) = z(\Delta) + \lambda\, z(s_{\text{probe}})$. Δ is the model's log-probability of the reading under which the conclusion is withdrawn minus that of the reading under which it stands, summed over tokens; s_{probe} is the probe's out-of-fold score; z standardizes over all items, ignoring labels. AUC is pooled over the 391 items in modus ponens form. The likelihood AUCs of Section 4.4 and this section average over tokens on all 782 items, so they differ from the values at λ=0. Columns are the three wordings of the two readings; sufficiency is the reported one. Fold split 1 assigns item groups to cross-validation folds as in Section 4.4; splits 2 and 3 reassign them with two seeds. The dashed line marks fold split 1's probe-only AUC. Intervals resample item groups.

answer and another answer: 21 of the 45 Countdown items, 13 of the 19 certified MuSR items and 54 of the 782 Belief-R rows. On Countdown, with a fresh pool of 64 chains per item, the Borda vote and majority voting each solve 1 item (paired difference 0.000 $[-0.067, +0.067]$), and a Borda vote over a uniformly random ranking solves 1.1 in expectation. On MuSR and Belief-R the Borda vote returns the majority answer on every item.

Self-Eval beam search. Self-Eval guided beam search (Xie et al., 2023) keeps 5 beam entries, samples 2 successors for each, and scores each step by combining its likelihood with the model's probability that the step is correct. We ran the published procedure with one seed. On Countdown, Self-Eval beam search solves 2 of the 45 items, against 14 for state decoding with the learned head at λ=0.5 alone; the 0.400 of Section 4.2 also counts items solved at λ=1. The paired difference summed over items is -12 $[-19, -6]$. On MuSR it fixes 3 of the 19 certified items, against 9 for likelihood scoring with model-written sentences over the enumerated support [ORACLE], and the summed difference is -6 $[-11, 0]$. A uniform guess over the answer options fixes 5.23 items in expectation. Two differences favor state decoding. On Countdown, state decoding accepts at each layer any state that equals the target, while Self-Eval beam search does not check the target during the search. On MuSR, the enumerated states come from the annotated moves, while Self-Eval beam search reads only the story, the question and the options. On MuSR the model's judgment of a step is nearly constant: on nine items outside the test set, it judges the step correct on 217 of 218 queries. Its

MuSR runs also had few candidates to choose from. After deduplication, 481 of its 555 nodes hold a single successor. On the 22 items (the 19 certified items and 3 knowledge controls), 20 runs are unfinished after 16 steps, and 7 end in a table whose queried entry matches no option and are scored as wrong. Belief-R was not run.

A* search with a process reward model. A* decoding (Chatziveroglou, 2025) repeatedly expands the partial solution of lowest cost. At each expansion it samples 16 successors from the model and scores each step with a mathematical PRM, Qwen2.5-Math-PRM-7B (Zhang et al., 2025). We ran the published procedure on Countdown with three seeds and read each seed separately. A* solves 3, 3 and 1 of the 45 items, against 14 for the reference above. The summed paired differences are -11 $[-18, -4]$, -11 $[-18, -4]$ and -13 $[-20, -7]$. A uniformly random legal move at each step solves 0.054 items in expectation. Besides the scoring, the two decoders differ in the search (best-first with backtracking, against a layered beam with merging), in the number of candidates per node (16 against 8) and in the target check, which favors state decoding as above. MuSR and Belief-R were not run.