Correct So Far, But Where Next? Diagnosing Next-Step Failures in Multi-Step Reasoning

Anonymous authors
Paper under double-blind review

Abstract

Chain-of-thought reasoning improves multi-step problem solving by decomposing a difficult computation into simpler steps, but decomposition only helps if the steps are composed correctly. When a reasoning chain first goes wrong after a sequence of correct steps, existing evaluations do not distinguish whether the model chose the wrong computation to perform next or chose the right one and executed it incorrectly. Meanwhile, verified, truncated, or relayed reasoning traces are reused in practice on the assumption that a correct intermediate result is a state from which the computation can continue. We show that this assumption can fail even when every supplied step is correct and the remaining computation is well within the model's ability. Across controlled serial tasks with a unique correct next step, models often preserve the correct intermediate result yet continue from the wrong place, repeating or skipping steps or taking the wrong branch. These failures primarily reflect choosing the wrong computation to perform next rather than executing the right one incorrectly. Explicitly identifying the next step or the remaining computation removes much of the gap, whereas merely restating the current result does not. Training and internal-state interventions reinforce this distinction: the same remaining computation can succeed or fail depending on how it is entered, and internal states formed by a fresh run can repair a failed continuation. A correct intermediate result is therefore not necessarily a reasoning state from which the model can reliably continue.

1 Introduction

Chain-of-thought prompting and scratchpads let a language model carry out a long computation as a sequence of short ones (Nye et al., 2021; Wei et al., 2022), and the written tokens provably extend the serial computation available to a fixed-depth transformer (Feng et al., 2023; Li et al., 2024; Merrill & Sabharwal, 2024). Decomposition, however, is only half of the mechanism: the steps must also be composed, so the result of one step has to lead the model into the computation that comes next.

That multi-step reasoning fails is well documented. Models answer subquestions they cannot compose (Press et al., 2023), errors accumulate along a computation graph (Dziri et al., 2023), per-step accuracy decays over long horizons even when the plan is supplied (Sinha et al., 2026; Samiei et al., 2025; Panda et al., 2026), and step-level verifiers are trained to locate the first incorrect step (Lightman et al., 2024; Zheng et al., 2025). These measurements report that a step is wrong; they do not say why. When a trajectory is correct up to some point and the next step is wrong, the model may have chosen the wrong computation to perform next, or it may have chosen the right computation and executed it incorrectly. Final-answer accuracy, and even the position of the first incorrect step, conflate the two.

Continuing a multi-step computation requires keeping track of two distinct pieces of information: the *value* reached so far and the *progress* through the computation, that is, what has already been completed and what should come next. This distinction separates two ways a first wrong step can arise. A model may retain the correct value but transition to the wrong next operation, or it may select the correct operation and execute it incorrectly; we refer to these as *transition* and *execution errors*. Telling them apart requires tasks in which the correct next operation is unique, so that the first

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.

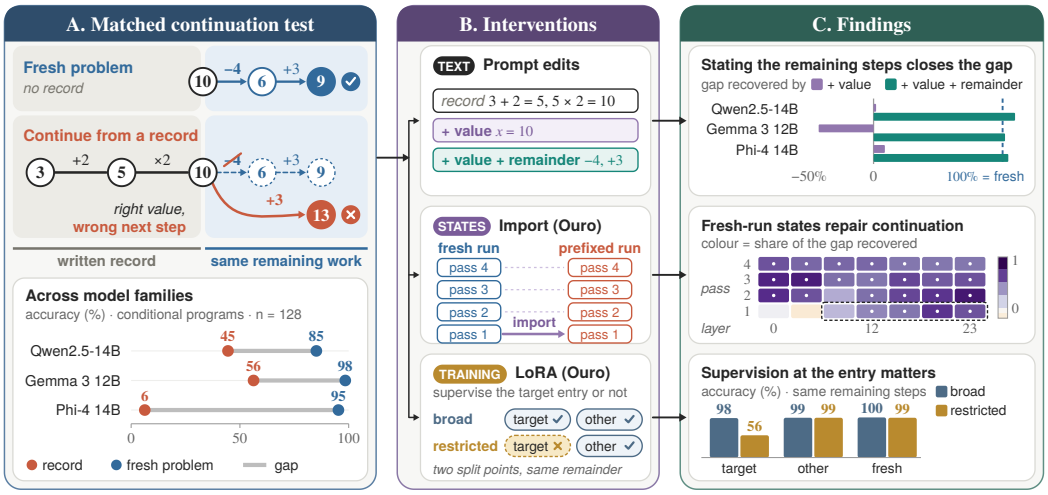

Figure 1: A correct record tells a model what has been computed, not what to compute next. (A) Schematic arithmetic example with an immediate answer: the fresh problem starts from $x = 10$ and applies -4, then $+3$; continuing from a correct record keeps $x = 10$, but its answer 13 implies that -4 was skipped. Below: accuracy on 128 conditional programs with two node visits left (ordinary wording); grey bars mark the gap. (B) The same items are probed with prompt edits, state imports in the looped Ouro-1.4B, and matched LoRA supervision. (C) Top: fraction of the gap recovered (ρ, Eq. 1) in conditional programs by adding the value, or the value and remaining rules. Middle: gap recovered by importing fresh-run states into a prefixed run at each pass and layer (Ouro-1.4B, symbolic arithmetic, four operations left), read at pass 4; dots mark intervals above zero. Bottom: strict token accuracy (Ouro-1.4B, mean of three seeds) at the target entry when it is supervised (broad) or withheld (restricted), at another supported entry into the same remainder, and on the fresh problem.

written action after a correct prefix can be compared with the correct one and a wrong final answer can be traced to the step at which the model entered; we use additive chains, conditional programs whose next node depends on the current state, and short Python programs. Two matched controls establish that the remaining work is within the model's ability: a *fresh problem* that asks for exactly the same remaining computation, and a one-step control that supplies the current value and the next operation.

Our main comparison holds the remaining computation fixed and varies only how the model arrives at the intermediate state: as a fresh problem that starts there, or by continuing from a correct written record of the completed steps, the form in which intermediate reasoning is verified, truncated, relayed, or reused in practice (Lightman et al., 2024; Lu et al., 2025; Wu et al., 2026; Setlur et al., 2026; Cemri et al., 2025). The same remaining work that a model solves as a fresh problem is lost when it is entered through a record of results, across model families, tasks, and split points, and in arithmetic whenever the remaining steps must be completed without being written (Figure 1). The loss lies neither in the value nor in execution: the current value is almost always retained and a single specified step is executed near ceiling, but the model enters the remaining computation at the wrong step, repeating completed operations, skipping pending ones, or entering the wrong branch. Naming the next step or the remaining operations removes most of the loss, whereas restating the value does not, and a record that states each successor is continued near ceiling, so what matters is whether the trace encodes how the computation should continue. Matched training comparisons that vary which split points receive target supervision show that entry into the same remainder succeeds from supervised positions and fails from withheld ones, and a looped language model serves as a mechanistic probe: its repeated passes through shared layers provide a time axis (Zhu et al., 2025b), and internal states imported from a fresh run repair the entry.

We make three contributions:

- **A formulation of continuation failures in multi-step reasoning** that separates the value reached from progress through the computation, and transition errors from execution errors.

- **A controlled diagnostic framework** for determining whether a correct intermediate state is usable for further computation, by matching continuation against fresh-problem and one-step controls on exact serial tasks.
- **Evidence that correctness of an intermediate result does not by itself guarantee correct continuation**, together with text, training, and internal-state interventions that identify missing next-step information as a central source of failure.

2 Related Work

2.1 Written steps as computational state

Chain-of-thought and scratchpads make intermediate computation part of the input for later predictions (Nye et al., 2021; Wei et al., 2022), and theory treats each written token as a recurrent state that extends serial computation (Feng et al., 2023; Li et al., 2024; Merrill & Sabharwal, 2024). Tokens that store intermediate results behave like program variables (Zhu et al., 2025a), and trace supervision can create written states that later computation demonstrably reads (Shih et al., 2026). State-tracking studies ask whether a model maintains the current state of an evolving world (Kim & Schuster, 2023; Prakash et al., 2024; Li et al., 2025; Zhang et al., 2025). These concern the value; we ask whether a correct value leads the model to the correct next operation.

2.2 Continuing from partial traces

Decomposition methods give each stage an intermediate answer and an explicit next subproblem (Zhou et al., 2023; Khot et al., 2023); a partial derivation describes only what has happened. Continuation from human-written partial solutions has been measured (Hendrycks et al., 2021); faithfulness studies truncate or edit traces (Lanham et al., 2023; Turpin et al., 2023); trace analyses resample continuations from prefixes (Bogdan et al., 2025; Ballon et al., 2026; Bulut, 2026) or perturb a model's own chain (von Recum et al., 2026). Reasoning Relay hands truncated chains to other models (Lu et al., 2025), a provider–receiver design finds that a prefix's value depends on task and receiver (Cheng et al., 2026), correct self-generated prefixes help on open mathematics (Sun et al., 2026), and prefix utility has been proposed as a training signal (Zhou et al., 2026). These comparisons measure sampled or perturbed prefixes against the full solution or a restart; we hold the prefix correct and the remainder fixed and compare records of results with the model's own trajectory, separating execution of the remainder from taking the correct next step.

2.3 Failures of multi-step execution

Models answer subquestions they cannot compose (Press et al., 2023), accumulate errors along a computation graph (Dziri et al., 2023), and follow locally valid but globally wrong steps (Saparov & He, 2023). With the plan supplied, long-horizon accuracy still decays, partly because a model conditions on its own earlier mistakes (Sinha et al., 2026); procedural benchmarks report dropped, duplicated, and under-executed steps (Samiei et al., 2025; Panda et al., 2026; Pai & Xian, 2026); failures in long traces begin at a few early transitions (Zhu et al., 2026); and irrelevant context or premise order change what is executed (Shi et al., 2023; Chen et al., 2024). Step skipping has also been studied as a learned efficiency (Liu et al., 2024); here it is an error. Our chains are short and the supplied steps correct, yet the failure appears at the first step after a record of results and depends on the split point and the form of the record rather than on length.

2.4 Mechanisms and training

Probes on synthetic grade-school problems show that a model represents which quantity to compute next before writing it (Ye et al., 2024); Cabannes et al. (2024) identify an iteration head that selects the next input through positional information; and trained transformers acquire positional shortcuts that fail outside their training distribution (Liu et al., 2023; Nikankin et al., 2025) or condition on the whole history rather than the current state (Phua et al., 2026). In latent multi-hop recall, an intermediate entity can be retrieved yet not used by the next hop (Yang et al., 2024; Biran et al., 2024); step-to-step information flow likewise breaks in failing long traces (Xu et al., 2026). Continuous and

recurrent reasoning compute in hidden states (Hao et al., 2024; Geiping et al., 2025; Zhu et al., 2025b), hybrid methods learn when to write steps (Lin et al., 2026; Yang et al., 2026; Fan et al., 2026), and Liang & Pan (2026) trace intermediate results inside latent reasoning; we use a looped model only as a probe.

3 Diagnosing the First Wrong Step in Multi-Step Computation

3.1 Transitions and two kinds of error

A serial problem consists of an initial state s_0 and operations $(o_1, \ldots, o_d)$, with $s_k = o_k(s_{k-1})$ and answer s_d. A trajectory is correct through step t when its written states $s_{0:t}$ are correct. The next step is a *transition*: from s_t, the model must apply o_{t+1}. We write $\hat{o}_{t+1}$ for the operation the model actually applies and $\hat{s}_t$ for the value it applies it to, both read from the written step. The step is a *transition error* when $(\hat{s}_t, \hat{o}_{t+1}) \neq (s_t, o_{t+1})$, and an *execution error* when the pair is correct but the written result differs from $o_{t+1}(s_t)$. Transition errors are described, not subdivided, by whether $\hat{o}_{t+1}$ is an operation already completed (a *repeat*), a later one (a *skip*), or, in a conditional program, a node off the correct path (a *jump*), and by whether the value $\hat{s}_t$ was retained.

We measure transitions in three regimes. When the model writes the whole trajectory, the *hazard* h_t is the probability that the first error occurs at step $t+1$ given that steps 1 to t are correct. When a correct prefix is supplied, we measure $P(\hat{o}_{t+1} = o_{t+1}, \hat{s}_t = s_t \mid s_{0:t})$ at every split point t, either from the first written action or, when the model answers immediately, from the answer: executing every suffix of the original operations from s_t identifies an *implied* split point $\hat{t}$ when the match is unique, with $\hat{t} < t$ a repeat and $\hat{t} > t$ a skip (Appendix G.2). Two controls certify that the remaining work is within capability. The *fresh problem* starts at s_t and asks for exactly the remainder $R_t = (o_{t+1}, \ldots, o_d)$, and the one-step control supplies s_t and o_{t+1} and asks for s_{t+1}. With A_{fresh} and A_{partial} the paired accuracies of the fresh problem and of continuation from the supplied prefix, we report

$$G = A_{\text{fresh}} - A_{\text{partial}}, \qquad \rho_I = \frac{A_I - A_{\text{partial}}}{A_{\text{fresh}} - A_{\text{partial}}}, \tag{1}$$

where ρ_I is the fraction of a positive gap recovered by intervention I. All items are retained, so fresh accuracy also records the difficulty of the remaining work. Separately, replacing s_t with a valid s' tests *value adoption*: whether the output becomes $\text{exec}(s', R_t)$, even though this need not solve the original problem. We write j for the number of written steps of a supplied prefix, also called the boundary.

3.2 Tasks, records, and writing formats

Addition and subtraction chains permit exact control over the current result and remaining instructions. In the *natural format*, the question describes the complete task and the partial chain stops at a computed result:

```
Q: Start with 12, subtract 2, subtract 5, subtract 4, add 1.
What is the result?
A: 12 - 2 = 10, 10 - 5 = 5. The result is
```

The matched fresh problem starts at 5 and asks only to subtract 4 and add 1. This immediate-answer readout requires the remaining steps to be completed without being written; a *written-continuation* variant instead ends the prompt after the last written step and lets the model write the remaining steps before the result. In the *symbolic format*, the original question and evaluated segments remain, but the expression `5 - 4 + 1 =` is supplied immediately before the answer, so execution is tested with the remaining instructions already explicit. Multiple written values are specified by a split set $S \subseteq \{1, \ldots, d-1\}$.

In conditional programs, each node tests the current integer, applies an update, and selects a successor. A supplied record lists completed transitions as results (`At X, x was 10 and becomes 8.`); a record in the model's own step format also names each branch outcome and successor. The fresh counterpart supplies the current node, value, and entire reachable subgraph. Node names and

Table 1: Factorial intervention with the original question fixed. Each cell gives the execution target predicted from the two supplied inputs. All four targets are distinct on the selected items.

Written value	Correct remainder R_j	Altered remainder $R_{j'}$
Correct s_j	exec(s_j, R_j)	exec$(s_j, R_{j'})$
Altered s'	exec(s', R_j)	exec$(s', R_{j'})$

rule order are randomized, and the model generates further transitions before answering. Multiplication chains, discrete heading or card states, and short Python programs provide additional comparisons (Section 6). Supplied prefixes are correct throughout; Appendix L evaluates generated prefixes.

3.3 Interventions at three levels

Text. With the original question fixed, *value only* replaces the partial derivation with an explicit statement of the current result; *value + remainder* adds the remaining operations; *chain + value + remainder* retains the derivation before giving the same information. Further controls re-ask the remaining problem or the original question, vary the history leading to a fixed value, and replace the chain with length-matched filler (Appendix A).

The factorial design independently alters the value and remainder (Table 1). An altered remainder repeats a consumed operation or skips the first remaining one. The four execution targets differ, so outputs identify which supplied inputs were followed and which were retained from the original question.

In conditional programs, a *pointer* names only the next node, retaining the full graph, start, and trace; value reminders, token-matched neutral text, and completed-visit counts control other effects of adding a cue (Appendices D and E).

Representations. We import a fresh run's residual states into a continuation run at specified positions, layers, and recurrent passes, then let the receiver finish; position-matched and different-item donors control absolute position and content (Section 5.2).

Training. LoRA regimes share the model, task construction, and optimization settings while varying supervised split points: omitting all interior boundaries tests a new type of entry, and withholding one interior cell tests interpolation. A matched design preserves the remaining computation, answer targets, and demonstrations across arms while varying which depth–boundary pairs receive target supervision, so supported and withheld entries into the same remainder test the effect of that supervision.

3.4 Evaluation protocol

Pretrained natural-format arithmetic uses a closed sequence score over answers 0–20; written continuations, standard-model symbolic comparisons, and discrete programs use greedy generation; conditional programs allow generated reasoning and score the final integer. First actions are read by a frozen, blind-audited extractor that records the node, input value, result, and successor of each asserted step (Appendix E); steps it cannot read are reported as indeterminate. Accuracy differences and intervals are paired over problems. Appendix A specifies all readouts, including strict token accuracy for matched supervision, and Appendix F the first-wrong-step protocol.

4 Experiments

4.1 Experimental setup

The behavioral study covers Qwen2.5-0.5B/1.5B/3B/7B and Qwen3.5-0.8B/2B/4B-Base. Conditional programs evaluate Qwen2.5-14B-Instruct, Gemma 3 12B-IT (Gemma Team, 2025), and Phi-4 14B (Abdin et al., 2024) with generated reasoning. Natural-format arithmetic evaluates every internal split at depths four to six, with 210 answer-balanced items per depth and eight demonstrations sharing no arithmetic step with their target; the first-wrong-step study uses 384 new programs of

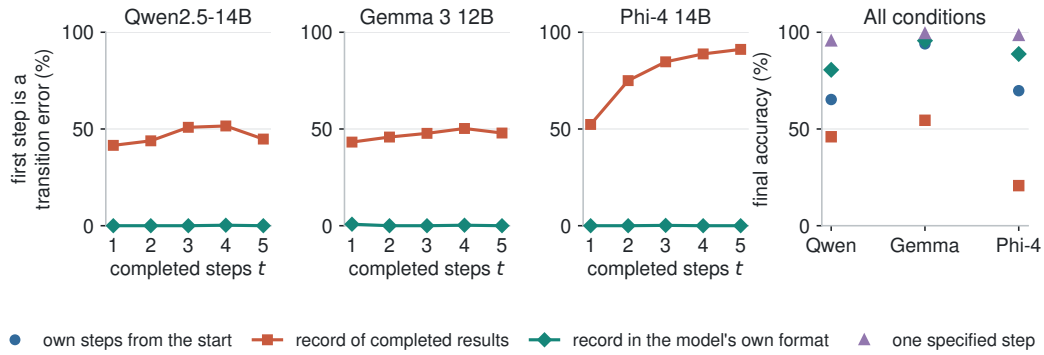

Figure 2: First-step transition errors by number of supplied steps. Share of first written steps that are transition errors after a record of completed results (squares) and after the same steps in the model's own format, which names each successor (diamonds); both wordings pooled, 192–768 outputs per point. Right: final accuracy under the four conditions (Table 6).

depth three to six and the same arithmetic items with written continuation (Appendix F). Weights are fixed, and the same items are evaluated in every condition of a comparison.

Ouro-1.4B and Ouro-2.6B provide recurrent probes with shared blocks of 24 and 48 layers (Zhu et al., 2025b), read at the fourth pass. Boundary-supervision experiments use Ouro-1.4B and Qwen2.5-3B with matched LoRA optimization across three seeds (Appendices A and I).

4.2 When does the next step go wrong?

Continuing from a written record of results produces wrong next steps. When correct steps are supplied as a record of results, the first written step is a transition error on 0.40–0.52 of prompts for Qwen2.5-14B, 0.39–0.54 for Gemma, and 0.73–0.74 for Phi-4, whereas execution errors at that step are at most 0.06. Final accuracy falls accordingly (Table 6, Figure 2), and the cell with two remaining visits reproduces Figure 4a. For Qwen and Gemma the rate is roughly flat in the number of supplied steps and the remaining length, whereas for Phi-4 it rises from about half of the prompts after one step to over nine in ten after five. A single specified step is executed near ceiling, so the loss is not one of execution. Adding the successor to every line of the same record, in the same position of the prompt, lowers the first-step transition share to 0.00–0.01 in all three families and raises final accuracy to 0.58–0.95, and writing the record in the model's own step format in the prompt does the same. In the independent run, one added sentence naming the next node lowers wrong first entries from 114, 81, 139, and 155 of 192 to 5, 0, 11, and 0 across its four models, whereas demonstrations that write the selected branch before each update leave them in place (Appendix F). The same holds for tabular and narrative records the model has never produced (0.46–0.89 without the successor, 0.00–0.01 with it) and for another model's trajectory (at most two wrong first steps in 788–1,238 prompts; Appendix F).

When the trace names the next step, the model takes it. A model's own trajectory names each successor, and when it writes the trajectory from the start, wrong next steps are uncommon. On arithmetic chains written step by step, four base models complete depths four to six with a per-step hazard of at most 0.01. On conditional programs (Table 6), the first error of Qwen2.5-14B and Gemma is never a step taken at a node other than the one the previous step named: it is made at a correctly entered node, and more often than a wrong update it is a wrong successor written after a correct update, on 0.21 of Qwen2.5-14B's trajectories (0.06–0.07 of steps at the first four positions), 0.04 of Gemma's, and 0.08 of Phi-4's readable ones. It then follows the node it named (221 of 222 cases) and mostly ends with a wrong answer (208). An independent four-model run that reads condition, value, and successor as separate fields finds the same error, and writing the selected branch before each update reduces it (Appendix F).

In arithmetic the loss appears only when the remaining steps are not written. With written continuation, the four base models continue a correct prefix with 0.99–1.00 accuracy at every split point; for the two larger models the first written step is correct on every item in every cell (210 per

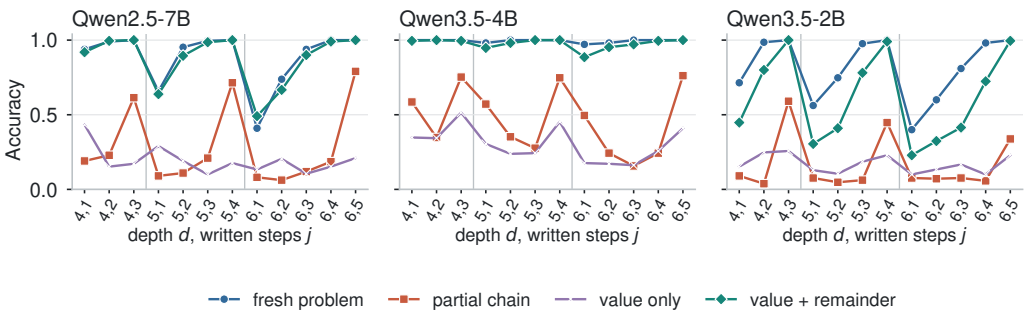

Figure 3: Immediate-answer continuation across models and split points. Natural-format accuracy for three base models with 210 paired items per cell and closed readout; d and j denote depth and completed steps. Stating the remainder recovers much of the gap to matched fresh problems, whereas restating the value alone does not. Figure 6 includes all models and chain-preserving controls.

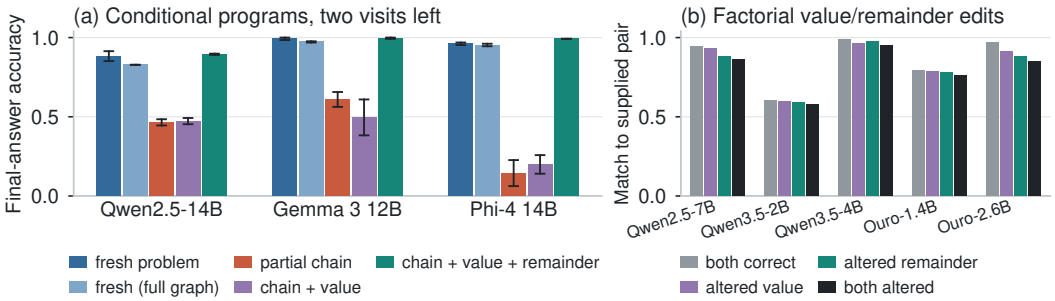

Figure 4: Conditional programs and factorial edits. (a) Final-answer accuracy on 128 conditional programs with two node visits left; bars average the ordinary and explicit wordings and ticks mark the two (Table 5). Full-graph fresh prompts retain every original rule. (b) Fraction of answers matching the supplied value–remainder pair, pooled over 1,122 item–split pairs per model (Table 3).

cell). Under the immediate-answer readout of the natural format, the same prefixes leave large gaps (Figure 3): at depth five, Qwen2.5-7B answers correctly on 0.09, 0.11, 0.21, and 0.71 of items after one to four written steps, against 0.66, 0.95, 1.00, and 1.00 on the matched fresh problems, and Qwen3.5-4B shows the same profile at depth six. Wrong answers traced to their implied split point mostly skip ahead after early splits and increasingly repeat completed operations as j grows, more so when only the value is stated (Figure 11); the gap closes only when a single step remains. Replacing the derivation with *value only* leaves a substantial loss, whereas adding the remaining operations brings accuracy close to the fresh reference, even when the completed chain stays in context. Under this readout a sentence naming only the next operation is executed as if it were all that remained (0.98–1.00 with one step left, 0.00–0.06 otherwise), counting completed steps does not help, and stating the remaining operations recovers most of the loss (0.62–0.98; Appendix F).

4.3 Transitions with conditional control flow

The loss persists when continuation includes generated reasoning and branch selection. We supply two correct node transitions as a record and let the model generate the rest; the matched fresh problem starts at the resulting node and value, with the same reachable rules. All three instruction-tuned families perform substantially better on fresh execution than on continuation (Figure 4a); repeating the current value leaves the separation, whereas stating the entry node and remaining graph recovers most of it. A full-graph fresh control that keeps every original rule remains well above continuation, so unused rules do not explain the cost of entering through the record. All programs, including unparsable outputs, remain in the comparison.

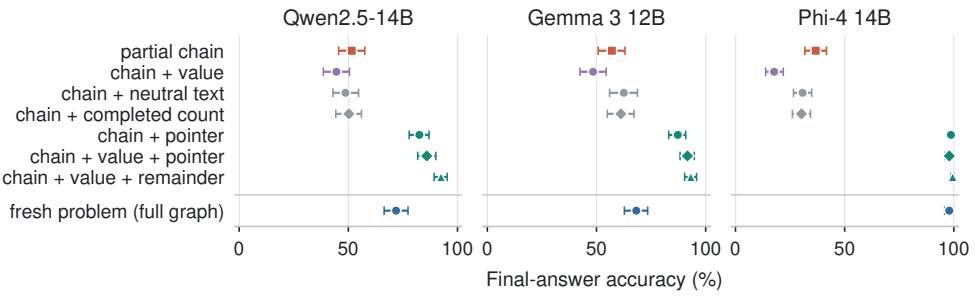

Figure 5: An entry pointer helps while the original program and trace remain. Accuracy on 192 programs, equally split over one, two, or three completed visits, with two visits left. Points average the two wordings; bars give paired 95% program-bootstrap intervals. Value + remainder repeats the rules after the same value and pointer. Fresh retains the full graph and has no history. All conditions share partial-chain demonstrations. Appendix D separates the wordings and lengths.

4.4 Independent interventions on value and remainder

Execution follows the supplied result and operations even when they conflict with the question. The factorial intervention in Figure 4b changes each input separately, then both. Models predominantly execute the supplied pair: an altered value changes the starting result, an altered remainder changes the operations applied to it, and changing both produces their composition; in Qwen2.5-7B and Ouro-2.6B an altered remainder carries some additional cost. Symbolic-format controls in Ouro confirm that a replacement value is executed and that changing the history behind a fixed value and remainder preserves most of the accuracy (Table 8, Figure 7).

5 Further Analysis

5.1 What fails at a wrong transition?

The value is retained and the step can be executed; the next operation is chosen wrongly. In the audited free continuations from a supplied record, a wrong first node accounts for 395 of the 456 incorrect answers, and 390 of these explicitly retain the correct current value; incorrect rule applications after a correct entry are a much smaller group (Figure 9). The one-step control shows that the operation itself is not the obstacle: given the current node and value, the three families execute a single step with accuracy 0.96, 0.99, and 0.98 (Table 6). Written in the model's own step format, which names each successor, the same prefix leaves essentially no first-step transition errors and final accuracy near that of the model's own trajectory.

Naming the next node helps without restating any rule. We retain the original start, full program, and correct trace, then add only a pointer to the next node. On new programs with different completed lengths, this improves continuation across all three families (Figure 5), whereas a neutral sentence of equal token length, a reminder of how many visits are complete, or a repeated current value have much smaller or inconsistent effects; repeating the reachable rules after value and pointer adds a further benefit mainly in Qwen. Even an intentionally wrong pointer often redirects the answer to the computation it specifies (Appendix D).

5.2 Where is the usable state formed?

A fresh run supplies useful computation before the recurrent solver finishes. We import fresh-run residual states into the corresponding remainder positions of a prefixed run, then read Ouro after its fourth pass. In Ouro-1.4B, imports from the middle of the first pass onward remove much of the continuation cost, earlier imports have little effect, and later passes retain the benefit (Figure 13); position-matched donors show the same pattern, reverse imports introduce a cost, and Ouro-2.6B recovers more strongly from its second pass.

The transferred state is specific to the donor's computation. Remainder-position states from another problem often make the receiver produce that problem's answer; at the answer position, later imports increasingly transmit the donor's answer, and applying the donor's boundary to the receiver's own value remains rare (Figure 14). Importing the broad training arm's answer-span states into the restricted arm on identical prompts improves strict token accuracy from the middle of the first pass and approaches donor accuracy by the next pass (Figure 15, Appendix H).

5.3 How does supervision at split points shape transitions?

Matched supervision changes continuation of the same remaining computation. A broad arm trains on six depth–boundary cells and a restricted arm on three; paired examples share the remainder, answer target, and demonstrations and differ only in which joint cells receive target supervision. At the withheld $(5, 2)$ entry in Ouro, restricted accuracy is 0.50–0.63, although both arms solve the same remainder fresh or through a supported entry at 0.98–1.00 (Table 10); most restricted-arm errors match a wrong contiguous window, often the one supervised at that depth.

The training effect also appears in a conventional Transformer, with substantial seed variation. Applying the same paired design to Qwen2.5-3B gives mean target-entry accuracies of 0.975 for broad and 0.571 for restricted supervision, while every run exceeds 0.95 on the matched fresh and supported-entry controls (Figure 17); one restricted run transfers well and two incur large losses. Re-asking the identical remainder as a fresh question brings all six runs close to ceiling, whereas a sentence-form reminder of the value and remaining operations is much less reliable, even for the broad arm (Appendix I.3).

Nearby supervised entries support interpolation. Omitting all interior target supervision produces systematic repetition or skipping, whereas withholding a single interior cell has a smaller effect (Figure 16, Appendix I).

6 Generalization

The contrast persists when operations interact and when the state is not a number. An internal multiplication by two or three makes the position of each update matter; with two operations left, Qwen2.5-7B and Ouro-1.4B move from partial-chain continuation to near fresh accuracy when the value and remainder are stated (Table 11). Heading and three-card programs with one operation left show the same gap across families, which state reminders leave and the remaining action closes in the larger models (Figure 19).

A program counter and variable frame do not uniformly remove the cost of using a trace. Short Python programs with a loop that revisits source lines keep the complete source in every prompt; fresh problems give the current variable frame and next line, and partial chains record the execution so far in event or statement form. All three families perform better on fresh execution under both renderings (Table 12), with the loss concentrated at the phases that require a jump. Giving the frame and next line nearly restores fresh accuracy in Phi-4 but has less consistent effects in Qwen and Gemma (Appendix K).

7 Discussion and Limitations

Local correctness does not ensure correct progression. A model's own steps carry the next operation along with each result; a record of results carries the results alone, and the model enters the remaining work at the wrong step while holding the right value. Verifying that the current step is correct therefore does not verify the transition to the next one, and a procedure that hands a verified, truncated, or relayed derivation to a solver should make the next step explicit or let the record name each successor; restating the full remainder after every step costs $O(d^2)$ tokens (Figure 21). Our evidence concerns short computations with a unique correct next operation; open-ended reasoning admits several valid next steps and requires equivalence classes over continuations. The position-dependent error pattern describes which step is entered, not an internal counter, and the Ouro interventions do not identify a portable progress variable.

AI USE STATEMENT

Generative AI tools were used in three ways. To aid and polish writing: language models edited and polished passages of text written by the authors, and all wording, claims, and numbers were checked by the authors against the frozen experimental records. For retrieval and discovery: language-model-based search was used to locate candidate related work, and every cited paper was read and verified by the authors. For research ideation and execution: language models assisted in writing experiment and analysis scripts, which the authors reviewed and tested for correctness against independent interpreters and audited counts. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Appendix A specifies task construction, prompt formats, readouts, and the evaluation protocol; Appendix F gives the first-wrong-step protocol, including seeds, decoding settings, and the classification rules; Appendix E describes the frozen extractor and the independent interpreter used to audit written traces; and the training and residual-state procedures are given in Appendices I and H. Every result is computed from frozen artifacts that record the generator, the displayed prompts, model revisions, raw generations, and analysis code with content hashes; these artifacts will be released upon publication.

ETHICS STATEMENT

This work evaluates publicly available language models on synthetic arithmetic and program-execution tasks. It involves no human subjects, personal data, or sensitive content, and we do not foresee direct harmful applications of the findings.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Xin Wang, Rachel Ward, Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang. Phi-4 technical report, 2024. URL https://arxiv.org/abs/2412.08905.

Marthe Ballon, Brecht Verbeken, Vincent Ginis, and Andres Algaba. Probing the Trajectories of Reasoning Traces in Large Language Models. *arXiv preprint arXiv:2601.23163*, 2026.

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. Hopping Too Late: Exploring the Limitations of Large Language Models on Multi-Hop Queries. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://arxiv.org/abs/2406.12775.

Paul C. Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. Thought Anchors: Which LLM Reasoning Steps Matter? *arXiv preprint arXiv:2506.19143*, 2025.

Yigit Utku Bulut. It's the Problem, Not the Path: Budget and Difficulty Confounds in LLM Reasoning Trajectories. *arXiv preprint arXiv:2609.03436*, 2026.

Vivien Cabannes, Charles Arnal, Wassim Bouaziz, Alice Yang, Francois Charton, and Julia Kempe. Iteration Head: A Mechanistic Study of Chain-of-Thought. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.02128.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why Do Multi-Agent LLM Systems Fail? *arXiv preprint arXiv:2503.13657*, 2025.

Xinyun Chen, Ryan A. Chi, Xuezhi Wang, and Denny Zhou. Premise Order Matters in Reasoning with Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.08939.

Xinyuan Cheng, Beiduo Chen, Philipp Mondorf, and Barbara Plank. Reasoning that Travels: Dissecting How Chain-of-Thought Transfers Across Models. *arXiv preprint arXiv:2605.28913*, 2026.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and Fate: Limits of Transformers on Compositionality. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.18654.

Ying Fan, Anej Svete, and Kangwook Lee. Bridging the Gap Between Latent and Explicit Reasoning with Looped Transformers. *arXiv preprint arXiv:2606.31779*, 2026.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards Revealing the Mystery behind Chain of Thought: A Theoretical Perspective. In *Advances in Neural Information Processing Systems*, 2023.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. In *Advances in Neural Information Processing Systems*, 2025.

Gemma Team. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training Large Language Models to Reason in a Continuous Latent Space. *arXiv preprint arXiv:2412.06769*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *Advances in Neural Information Processing Systems: Datasets and Benchmarks Track*, 2021. URL https://arxiv.org/abs/2103.03874.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed Prompting: A Modular Approach for Solving Complex Tasks. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.02406.

Najoung Kim and Sebastian Schuster. Entity Tracking in Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023. URL https://aclanthology.org/2023.acl-long.213/.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, Saurav Kadavath, Shannon Yang, Thomas Henighan, Timothy Maxwell, Timothy Telleen-Lawton, Tristan Hume, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Measuring Faithfulness in Chain-of-Thought Reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

Belinda Z. Li, Zifan Carl Guo, and Jacob Andreas. (How) Do Language Models Track State? In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2503.02854.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of Thought Empowers Transformers to Solve Inherently Serial Problems. In *International Conference on Learning Representations*, 2024.

Jia Liang and Liangming Pan. Do Latent-CoT Models Think Step-by-Step? A Mechanistic Study on Sequential Reasoning Tasks. *arXiv preprint arXiv:2602.00449*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Hanyu Lin, Min Cai, Jiawei Wen, and Haodi Zhang. Tyler: Typed Latent Reasoning for Language Models – When to Think, What to Compute, and How Much to Allocate. *arXiv preprint arXiv:2606.16360*, 2026.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers Learn Shortcuts to Automata. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=De4FYqjFueZ.

Tengxiao Liu, Qipeng Guo, Xiangkun Hu, Cheng Jiayang, Yue Zhang, Xipeng Qiu, and Zheng Zhang. Can Language Models Learn to Skip Steps? In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2411.01855.

Leo Lu, Jonathan Zhang, Sean Chua, Spencer Kim, Kevin Zhu, Sean O'Brien, and Vasu Sharma. Reasoning Relay: Evaluating Stability and Interchangeability of Large Language Models in Mathematical Reasoning. *arXiv preprint arXiv:2512.20647*, 2025. URL https://arxiv.org/abs/2512.20647. NeurIPS 2025 Workshop on Socially Responsible and Trustworthy Foundation Models.

William Merrill and Ashish Sabharwal. The Expressive Power of Transformers with Chain of Thought. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.07923.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic Without Algorithms: Language Models Solve Math with a Bag of Heuristics. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.21272.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show Your Work: Scratchpads for Intermediate Computation with Language Models. *arXiv preprint arXiv:2112.00114*, 2021.

Dheeraj Mohandas Pai and Lu Xian. Long-Horizon State Tracking in LLMs: Executing MD5 through a Deep Sequence of Dependent Tool Calls. *arXiv preprint arXiv:2609.00012*, 2026.

Sailesh Panda, Pritam Kadasi, Abhishek Upperwal, and Mayank Singh. When LLMs Stop Following Steps: A Diagnostic Study of Procedural Execution in Language Models. *arXiv preprint arXiv:2605.00817*, 2026.

Yin Jun Phua, Tony Ribeiro, Tuan Nguyen, and Katsumi Inoue. Can Transformers Learn to Verify During Backtracking Search? *arXiv preprint arXiv:2605.22221*, 2026.

Nikhil Prakash, Tamar Rott Shaham, Tal Haklay, Yonatan Belinkov, and David Bau. Fine-Tuning Enhances Existing Mechanisms: A Case Study on Entity Tracking. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2402.14811.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and Narrowing the Compositionality Gap in Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023. URL https://aclanthology.org/2023.findings-emnlp.378/.

Mahdi Samiei, Mahdi Mansouri, and Mahdieh Soleymani Baghshah. The Illusion of Procedural Reasoning: Measuring Long-Horizon FSM Execution in LLMs. *arXiv preprint arXiv:2511.14777*, 2025.

Abulhair Saparov and He He. Language Models Are Greedy Reasoners: A Systematic Formal Analysis of Chain-of-Thought. In *International Conference on Learning Representations*, 2023.

Amrith Setlur, Zijian Wang, Andrew Cohen, Paria Rashidinejad, and Sang Michael Xie. Reuse Your FLOPs: Scaling RL on Hard Problems by Conditioning on Very Off-Policy Prefixes. *arXiv preprint arXiv:2601.18795*, 2026.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large Language Models Can Be Easily Distracted by Irrelevant Context. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. URL https://proceedings.mlr.press/v202/shi23a.html.

Benjamin Shih, John Winnicki, and Eric Darve. Do Models Read What They Write? Causal Registers in Scratchpad Reasoning. *arXiv preprint arXiv:2606.29522*, 2026. URL https://arxiv.org/abs/2606.29522v1. Version 1.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The Illusion of Diminishing Returns: Measuring Long Horizon Execution in LLMs. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.09677.

Yiliu Sun, Zicheng Zhao, Yang Wei, Yanfang Zhang, and Chen Gong. Well Begun, Half Done: Reinforcement Learning with Prefix Optimization for LLM Reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. URL https://arxiv.org/abs/2512.15274.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting. In *Advances in Neural Information Processing Systems*, 2023.

Alexander von Recum, Leander Girrbach, and Zeynep Akata. Are Reasoning LLMs Robust to Interventions on Their Chain-of-Thought? In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2602.07470.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Tsung-Han Wu, Mihran Miroyan, David M. Chan, Trevor Darrell, Narges Norouzi, and Joseph E. Gonzalez. Are Large Reasoning Models Interruptible? In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2510.11713.

Xiaoyu Xu, Yulan Pan, Xiaosong Yuan, Zhihong Shen, Minghao Su, Yuanhao Su, and Xiaofeng Zhang. Reasoning Fails Where Step Flow Breaks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://arxiv.org/abs/2604.06695.

Jiayu Yang, Chao Chen, Shengen Wu, Yinhong Liu, Yuxuan Fan, Lujundong Li, Songning Lai, Chengwei Qin, and Zhijiang Guo. Demystifying Hidden-State Recurrence: Switchable Latent Reasoning with On-Policy Reinforcement Learning. *arXiv preprint arXiv:2606.13106*, 2026.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. Do Large Language Models Latently Perform Multi-Hop Reasoning? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. URL https://arxiv.org/abs/2402.16837.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of Language Models: Part 2.1, Grade-School Math and the Hidden Reasoning Process. *arXiv preprint arXiv:2407.20311*, 2024.

Yifan Zhang, Wenyu Du, Dongming Jin, Jie Fu, and Zhi Jin. Finite State Automata Inside Transformers with Chain-of-Thought: A Mechanistic Study on State Tracking. *arXiv preprint arXiv:2502.20129*, 2025.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: Identifying Process Errors in Mathematical Reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025. URL https://arxiv.org/abs/2412.06559.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *International Conference on Learning Representations*, 2023.

Yuhang Zhou, Yixin Cao, and Guangnan Ye. From Correctness to Utility: Gain-Based Prefix Evaluation for LLM Reasoning. *arXiv preprint arXiv:2606.07190*, 2026.

Fangwei Zhu, Peiyi Wang, and Zhifang Sui. Chain-of-Thought Tokens are Computer Program Variables. *arXiv preprint arXiv:2505.04955*, 2025a.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling Latent Reasoning via Looped Language Models. *arXiv preprint arXiv:2510.25741*, 2025b.

Wei Zhu, Jian Zhang, Lixing Yu, Kun Yue, and Zhiwen Tang. Dissecting Failure Dynamics in Large Language Model Reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://arxiv.org/abs/2604.14528.

A Task construction and evaluation

A.1 Arithmetic items and prompts

Additive chains have depths one to six and keep every intermediate value in 0–20. Natural-format comparisons use an answer-balanced subset of 210 items per depth, with ten items for each final answer. All internal split points at depths four to six are evaluated. Each prompt has eight demonstrations in the target format, with no arithmetic step shared between demonstrations and target. Demonstration split points vary in the natural format; symbolic demonstrations match the pattern of supplied values. Table 2 shows the literal conditions, using one shared question and partial derivation.

Table 2: Natural-format prompt conditions. Shared question: "Start with 12, subtract 2, subtract 5, subtract 4, add 1. What is the result?" Here C expands to the exact partial derivation `12 - 2 = 10, 10 - 5 = 5.` Other than the fresh problem, each row follows this original question.

Condition	Target text
fresh problem	`Q: Start with 5, subtract 4, add 1. What is the result? A:`
partial chain	`A:` C `The result is`
value only	`A: So far the value is 5. The result is`
value + remainder	`A: So far the value is 5. Remaining: subtract 4, add 1. The result is`
chain + value	`A:` C `So far the value is 5. The result is`
chain + value + remainder	`A:` C `So far the value is 5. Remaining: subtract 4, add 1. The result is`
remainder re-asked	`A:` C, then a new line with the fresh question above
filler	`A:` length-matched filler `The result is`

In the symbolic format, evaluated segments precede the final unevaluated expression, for example `12 - 2 - 5 = 5, 5 - 4 + 1 = 2, 2 + 3 - 2 =`. The set of written split points varies over individual boundaries, pairs of boundaries, and a value after every operation. A question-only control retains the full task and the final expression while removing evaluated segments. Re-asking inserts either the fresh remainder question or the original full question immediately before the final expression. Figure 20 summarizes the supplied-state sweep.

A.2 Readouts and uncertainty

The closed readout selects the integer in 0–20 with the highest teacher-forced log-probability, including its terminator. The implementation evaluates candidate first tokens, second tokens for two-digit candidates, and the terminator. Ouro supplies a readout at each recurrent pass; pass four is the principal comparison. On the comparisons evaluated with both readouts, greedy and closed accuracy differ by at most 0.03. Pretrained natural-format arithmetic, Ouro's supplied-state sweeps and fresh-remainder imports, and the R1/R1h/W comparisons use the closed score. Matched-supervision controls and answer-span imports instead require every answer token, including the newline, to be the greedy argmax under teacher forcing; we call this strict token accuracy. Standard-model symbolic comparisons and discrete-state programs use greedy answers. Qwen factorial comparisons use the first integer generated within four tokens, with the closed score as a fallback when parsing fails. The generated-state sweep uses greedy generation and a parsed final integer (Appendix L).

Accuracy differences pair predictions on the same item. Symbolic and context comparisons use 2,000 bootstrap resamples; fresh-remainder import intervals use 1,000. Training summaries specify seed ranges separately. The original symbolic sweep defines equivalence by a paired difference within ± 0.03 and an interval within ± 0.05; substitution compares a smaller set of supplied split points with a larger set within the trained recurrent range. The fresh-chain reference uses a separate chain of the same remaining length. The main conditional-program comparisons permit generated steps and use the marked final answer, with 10,000 paired bootstrap draws (Appendix C). Their prompt profiles and decision rules are fixed before confirmation evaluation.

A.3 Compute and reproducibility

The original arithmetic, discrete-program, training, and residual-intervention experiments used approximately 200 GPU-hours on NVIDIA H100 and RTX 4090 hardware. New conditional-program evaluations use single H100 GPUs. Their frozen artifacts contain the generator, displayed prompts, model revisions, raw generations, and analysis code. All figures are generated from saved results, with item counts and readout conventions preserved. Supplementary summaries present the comparisons needed to interpret the claims; the associated machine-readable artifacts retain the individual predictions and full configuration tables.

B Additional behavioral comparisons

B.1 Natural-format continuation

Figure 6 separates fresh-solving ability from the cost of entering through a trace. It shows every model–depth–split cell, then the two controlled gains from stating the remainder: *value + remainder* versus *value only*, and *chain + value + remainder* versus *chain + value*. Both keep the current value explicit. The latter also retains the completed derivation. These gains are absolute accuracy differences; they remain interpretable when the fresh baseline is too low to support a normalized recovery fraction.

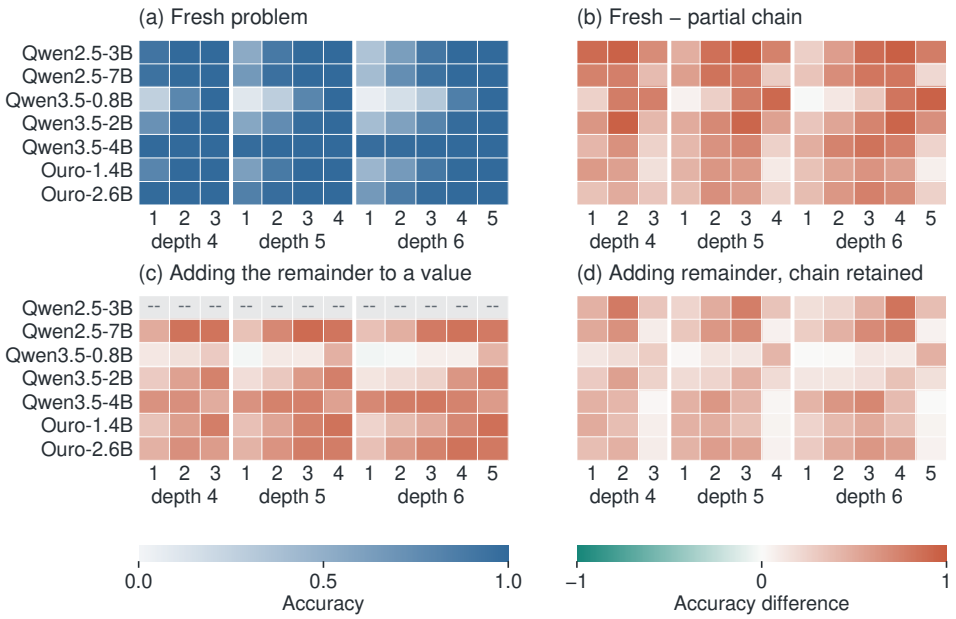

Figure 6: Fresh capability and remainder-description gains across all natural-format cells. Columns give completed operations j, grouped by total depth. (a) Fresh accuracy. (b) Fresh minus partial-chain accuracy. (c) Value + remainder minus value only. (d) Chain + value + remainder minus chain + value. All cells have 210 paired items and closed readout. Dashes mark the absent Qwen2.5-3B value + remainder conditions.

B.2 Factorial execution targets

The original question stays fixed while its supplied current value and remainder are varied. Altered boundaries first use $j' = j - 1$, repeating a consumed operation, with $j' = j + 1$ used for a forward shift. Replacement values are drawn from 1–19, requiring both programs to remain within 0–20 and all four final targets to differ. Selected cells contain 178–193 items. Six depth–split cells yield 1,122 item–split pairs from 628 distinct problem IDs per model.

Figure 7 shows every cell alongside the pooled rates in Table 3. In the both-altered condition, retaining the original value alone yields exec$(s_j, R_{j'})$; retaining the original remainder alone yields exec(s', R_j); retaining both yields exec(s_j, R_j). Distinct targets make these categories mutually exclusive. This decomposition measures which input pair determines the final computation without requiring a separate internal representation for each input.

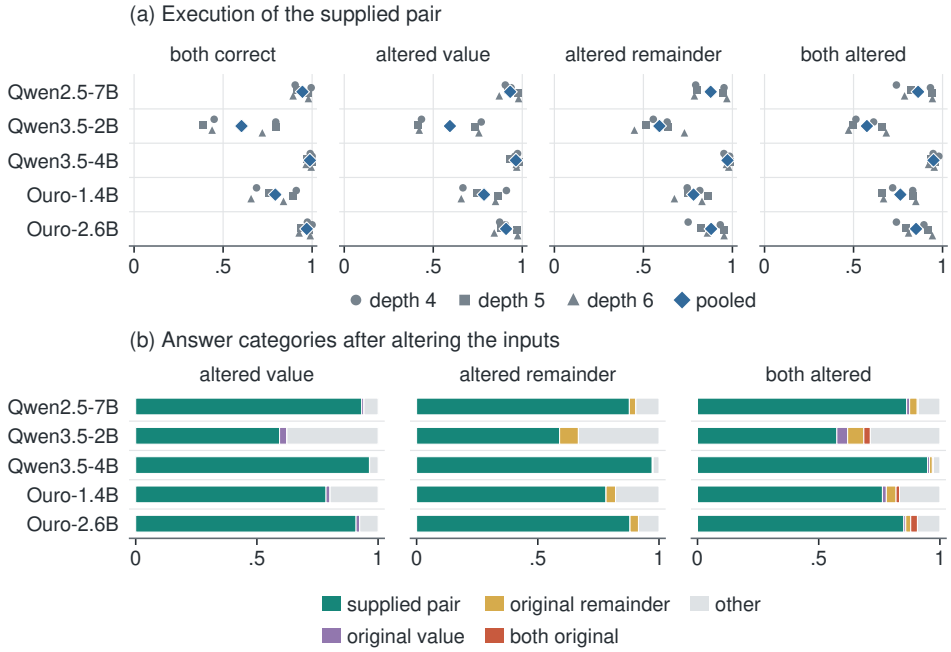

Figure 7: Factorial outcomes by cell and execution target. (a) Small markers show all 120 model–condition–split accuracies; diamonds give pooled rates weighted by item count. (b) Altered-input conditions decompose into mutually exclusive answers. "Original value" or "original remainder" retains only that input; "both original" retains both. Each model contributes 1,122 item–split pairs from 628 distinct problems. Ouro uses fourth-pass closed readout; Qwen uses the first generated integer within four tokens, with closed fallback.

Table 3: Execution of independently supplied values and remainders. Fraction of answers matching the supplied pair, pooled over 1,122 item–split pairs from 628 distinct problems per model. Qwen uses greedy answers with closed fallback; Ouro uses its fourth-pass closed score. Figure 7 separates alternative targets and individual cells.

Model	both correct	altered value	altered remainder	both altered
Qwen2.5-7B	0.946	0.932	0.878	0.863
Qwen3.5-2B	0.603	0.594	0.590	0.575
Qwen3.5-4B	0.988	0.964	0.972	0.949
Ouro-1.4B	0.793	0.785	0.782	0.763
Ouro-2.6B	0.970	0.909	0.881	0.851

C Conditional programs and generated continuation

C.1 Program construction and exact matching

Each item is a layered directed acyclic graph. A node tests $x < t_u$, selects an integer update, and moves to one of two successors; the terminal successor is `STOP`. Every nonterminal level has two distinct possible successors, so the executed path depends on the current state. A program with actual

path depth d contains $2d-1$ nodes, with randomized names and listing order. Thresholds range from 5 to 15, nonzero updates from -4 to 4, and initial values from 3 to 17. Accepted actual paths remain in 0–20 with distinct intermediate values.

The supplied trace records two executed nodes and their input and output values. The next node is determined by the final recorded input and its branch rule. The fresh problem explicitly gives this current node and value together with the reachable subgraph, including unexecuted alternative branches. Its displayed rules are identical to those in the remainder restatement. An independent text interpreter checks every transition and verifies that the fresh and partial conditions have the same remaining execution and answer.

C.2 Conditions, wording, and answer extraction

The partial-chain condition gives the full graph, original start state, and `Execution so far:` with the correct trace. A value reminder adds `The current value is x =` and the state. A remainder restatement adds `Resume at` the current node and state followed by `Remaining program:` with the exact fresh graph. The full-graph fresh control preserves every original rule but supplies the current node and value without a partial trace.

Two wordings are fixed in advance. Ordinary prompts ask for execution. The explicit cue adds: "The shown execution is correct and already complete up to its last line. Continue after it without repeating completed nodes." Fresh prompts receive the corresponding instruction that their start state is current. Both use two independent demonstrations. The main comparisons allow 384 generated tokens and demonstrate remaining transitions followed by `Final x: N`. Greedy decoding uses each instruction model's released chat template. Scoring requires a marked final integer; cosmetic Markdown and LaTeX delimiters are removed before applying the same extraction rule to every model and condition. Outputs without a valid final integer count as incorrect.

The formatting rule was fixed from fresh-only pilot outputs before the new-model confirmation sets were evaluated. It changes no Qwen2.5-14B or Gemma confirmation predictions. Raw generations and the strict, unnormalized scores are retained alongside the normalized readout. Thus the comparison measures the final stated answer without treating mathematical typesetting as execution failure. Table 5 uses this uniform rule.

Qwen and Gemma produce valid final integers in all confirmation outputs and never reach the token limit. Phi-4 has 15 invalid outputs across its 1,280 condition–program evaluations; four reach the token limit. Its ordinary partial-chain condition parses on 127 of 128 programs, and its explicit-cue partial-chain condition parses on all 128. The large continuation losses therefore remain among outputs with valid final answers. Formatting normalization changes the extracted answer for 158 Phi-4 outputs; both normalized and strict results are available for every condition in the numerical artifact.

C.3 Fresh-only pilots and held-out evaluation

Demonstration, pilot, and confirmation programs use disjoint fixed seeds and are hashed before evaluation. Pilots contain 16 programs per candidate horizon. A horizon qualifies when fresh accuracy is at least 0.70 and output validity at least 0.90 under both wordings. The main comparison uses two remaining visits and 128 independent confirmation programs, paired across all five conditions and both wordings. New families use the same frozen confirmation set and two-visit horizon as Qwen2.5-14B, conditional on passing their own fresh-only pilot. Confirmation items are not filtered by their predictions.

Complementary immediate-answer profiles in Qwen3.5-4B-Base and Qwen2.5-14B-Instruct allow 12 new tokens and score a leading integer, with no closed fallback. Their fresh-only pilots cover one to three remaining visits and select one. Table 4 gives the full comparison at this shorter horizon. These independently frozen sets have 160 programs per profile. The generated-continuation setting in the main paper tests two further state-dependent transitions.

Table 4: Immediate-answer continuation with one node visit remaining. Each wording uses 160 paired programs; invalid outputs count as incorrect. The full-graph fresh condition retains all original rules. All these outputs contain a valid integer.

Wording	fresh problem	fresh (full graph)	partial chain	chain + value	chain + value + remainder
Qwen3.5-4B-Base					
Ordinary	0.850	0.575	0.350	0.381	0.988
Explicit cue	0.812	0.388	0.406	0.375	0.981
Qwen2.5-14B-Instruct					
Ordinary	0.938	0.594	0.244	0.312	0.906
Explicit cue	0.894	0.662	0.356	0.319	0.887

C.4 A concrete entry error

One confirmation program starts at E with $x = 5$. The correct trace records E changing x to 9, then H changing it to 10. H's rule sends this execution to X. The fresh formulation executes X and Z to obtain 14. Given the partial trace, Qwen2.5-14B instead starts at P with the correct value 10, follows G and Z, and answers 12. Stating the current node and remaining graph restores the X–Z path and answer 14. All confirmation items contribute to the aggregate results.

A fixed-ID inspection of Phi-4 selects the first three parsed but incorrect partial-chain outputs under each wording. In all six, the model uses the correct current value but repeats the last completed node, and the displayed subsequent path yields its stated wrong answer. An independent program interpreter verifies these executions. This inspection illustrates a continuation error; it does not estimate its prevalence over all wrong answers.

Table 5: Conditional-program continuation across model families. Two node visits remain, with the same 128 programs per wording and greedy generated reasoning. Full-graph fresh prompts retain every original rule and explicitly give the current node and value. The explicit cue says completed nodes should not be repeated. Invalid outputs count as incorrect.

Model	Wording	fresh problem	fresh (full graph)	partial chain	chain + value	chain + value + remainder
Qwen2.5-14B	Ordinary	0.852	0.828	0.445	0.453	0.891
	Explicit	0.914	0.828	0.484	0.492	0.898
Gemma 3 12B	Ordinary	0.984	0.977	0.562	0.383	0.992
	Explicit	1.000	0.969	0.656	0.609	1.000
Phi-4 14B	Ordinary	0.953	0.961	0.062	0.141	0.992
	Explicit	0.969	0.945	0.227	0.258	0.992

D Entry pointers with the original program retained

D.1 Matched programs and prompts

The pointer study uses 192 new conditional programs, with 64 each at one, two, and three completed node visits. Every program has two visits remaining. The original graphs contain five, seven, or nine nodes, respectively. As in the earlier branching study, each node tests the current integer, applies an update, and selects a successor; node labels and rule order are randomized. The original execution stays within the task's integer range and has distinct intermediate values. Exact graph identities and identities invariant to node renaming and display order are disjoint from the earlier demonstrations, pilots, and confirmation programs.

All conditions share three worked demonstrations, one per completed length, in the unaugmented partial-chain format. Only the target prompt changes. This differs from the earlier study's condition-specific demonstrations, so the two studies provide separate within-study comparisons. Both ordinary

and explicit-instruction wordings are retained. The full-graph fresh condition keeps every original rule and starts at the current node and value. Every history-bearing condition instead retains the original start, full graph, and correct written trace.

The pointer adds only `The next node is U.`, with the appropriate node label. A value reminder adds `The current value is x = V.`; their combination appends these same two sentences. The remainder condition uses the identical combined reminder and then repeats the reachable subgraph. Consequently, the comparison between value plus pointer and value plus remainder changes the repeated instructions while holding their stated entry fixed. A neutral sentence, `The note style is plain.`, matches the pointer's token length in all three tokenizers. A separate reminder states the completed visit count without naming the next node. All reminders occur at the same position before generation.

A counterfactual pointer names the other node at the correct entry's depth. It preserves the current value, full graph, and trace, and still requires two visits. Its answer is computed independently before inference. The 15 programs whose counterfactual and original answers coincide remain in overall accuracy; pointer-following categories distinguish the 177 programs with different targets. A matching answer in a collision case cannot identify which pointer the model used.

D.2 Execution and analysis

The models, official chat templates, greedy decoding, 384-token limit, and formatting-only final-answer parser follow the earlier branching comparison. Invalid outputs count as incorrect. Twenty-four disjoint fresh-only pilot programs check execution and output handling; they do not select the confirmation programs, history lengths, or conditions. An independent interpreter replays the displayed rules and verifies every correct trace and both pointer targets. Prompt hashes retain the exact rendered inputs.

We report both wordings and each completed length. The aggregate comparison weights the three lengths equally. Paired intervals resample program identities within each length, keeping conditions paired; the two wordings of one program are not independent samples. The principal comparisons are pointer versus partial chain, pointer versus neutral text, value plus pointer versus value, and remainder restatement versus value plus pointer. Wrong-pointer following uses the separately identified counterfactual targets.

Figure 8 retains every wording and completed length. Fresh accuracy in Qwen and Gemma is lower than in the earlier experiment. The studies differ in both completed lengths and demonstration format: this study uses shared partial-chain demonstrations, while the earlier study uses condition-specific demonstrations. The graph and target matching remain exact; the input comparisons use conditions that retain the same history. Figure 5 reports absolute accuracies without normalizing them by a fresh gap.

E Diagnosing written execution traces

E.1 Extracting committed actions

We analyze the already generated branching continuations without requesting a new output format. The extractor receives only the response and its supplied history, with no model identity, expected next node, final-answer correctness, or gold execution. It records source spans for asserted node visits, input values, updated values, successors, and condition outcomes. A node mention alone does not establish execution: a committed numerical update or selected successor must accompany it. Quoted alternatives, deliberation, and recaps of supplied steps are distinguished from newly asserted actions. Unresolved revisions or incomplete statements remain indeterminate; missing values are not filled from the program or the final answer.

A separate interpreter executes the literal program rules and verifies the supplied history. It compares the extracted first action with the correct entry, then distinguishes a wrong node, a wrong input value at the correct node, and an incorrect rule application from a correct node and value. If the complete first transition is correct, later explicit errors are recorded separately. A fully correct written path

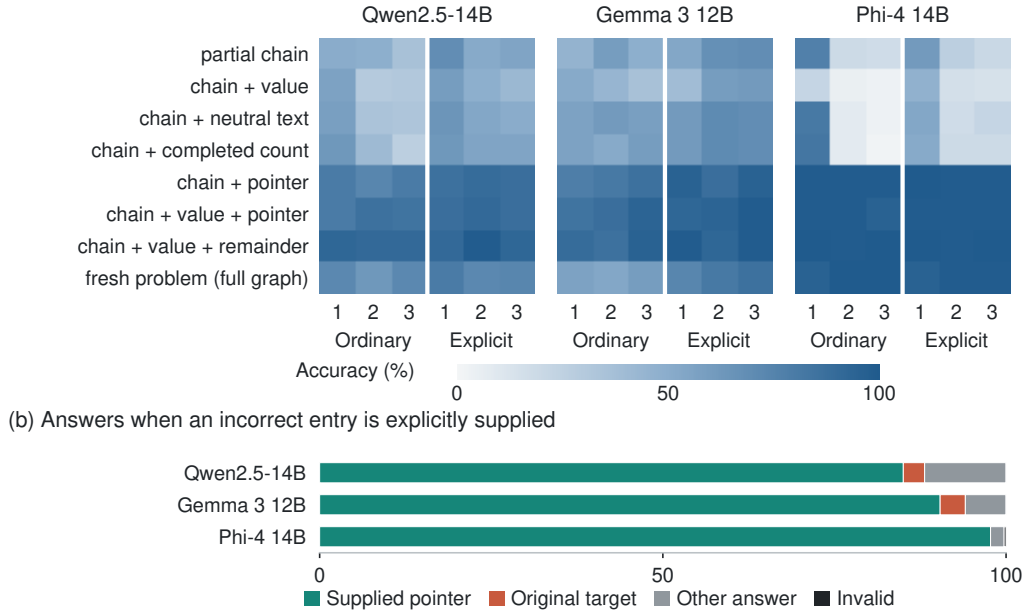

Figure 8: Pointer effects across prompt variants and altered entries. (a) Accuracy for all cells; columns give completed visits, with 64 programs per cell. (b) Final-answer categories after supplying the correct value and a wrong next node. The 177 programs with distinct original and altered targets give 354 outputs per model across both wordings. The 15 target-collision programs remain in the full accuracy results but cannot identify which pointer was followed.

can still disagree with the final-answer readout; this is kept distinct from an execution error. These categories concern the model's written actions, not its unobserved internal computation.

E.2 Coverage and validation

All 3,840 earlier confirmation responses remain in the analysis. The final-answer endpoint is the original frozen parser, including invalid and truncated responses. We report trajectory coverage separately from first-node coverage: a response may establish its entry without specifying enough detail to validate the whole path. For an entry-error count w among n responses, with u indeterminate entries, $[w/n, (w+u)/n]$ gives the range consistent with missing textual evidence. This is an identification range conditional on extraction validity, not a confidence interval.

The extraction audit uses 86 responses from 58 programs, sampled by a fixed hash within model, condition, and final-correctness strata. Review packets conceal those labels, expected executions, and parser outputs. Independent language-model reviewers transcribe the asserted transitions before joining the gold interpreter. A second review covers a fixed subset of 12 packets. Syntax-development examples are disclosed separately and excluded from this audit. All corrections to extraction or comparison rules retain their earlier versions and are applied to the complete corpus. The frozen extractor also analyzes the pointer-study continuations, with unsupported forms retained as indeterminate. Full diagnosis is possible for 8,586 of 10,368 outputs; first-node coverage is higher. A separate blinded check of 27 responses agrees on all 23 decidable first-node annotations. These coverage differences are kept separate from final-answer accuracy.

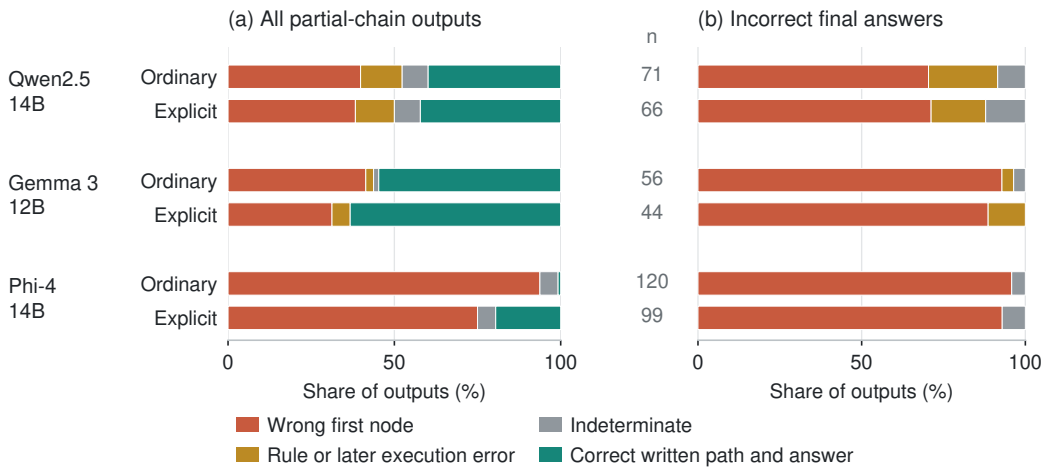

Figure 9: Wrong entries dominate the observable errors in free partial-chain continuations. Left: every partial-chain output, 128 programs per model and wording. Right: only incorrect final answers, with denominators shown. Wrong-node classifications can occur even when the final answer is coincidentally correct. Gray retains insufficient or conflicting textual evidence. Counts concern generated actions and are pooled over outputs, without treating repeated wordings as independent programs.

F First-wrong-step protocol

F.1 Conditional programs

The first-wrong-step study uses 384 new conditional programs, 96 at each depth from three to six, generated with the earlier layered generator under seed 2026092501; their name- and order-invariant identities are disjoint from every program used in the earlier demonstrations, pilots, and confirmation sets. A program of depth d visits d nodes before `STOP`, keeps every value in 0–20, and has distinct

intermediate values. Four conditions share the program, the system instruction, and the two wordings of the earlier study. In the *own trajectory* condition the prompt gives the program and the start state and the model executes from the beginning; three worked demonstrations of complete executions precede the target (seed 2026092502). In the *record* condition the correct prefix of t transitions, for every t from one to $d-1$, is listed after `Execution so far:` as `At X, x was a and becomes b.`, with the three frozen partial-chain demonstrations of the earlier pointer study. The *own-format record* condition places the same t transitions in the assistant turn, written exactly as the model's own demonstration lines (`At X, a < c is true; x becomes b; next is Y.`), so that each successor is named and generation continues from there. The *one-step* condition gives the program, the current node, and the current value, asks for a single step, and uses three one-step demonstrations (seed 2026092503); every position of every program is tested. Prompts are rendered with each model's chat template, frozen with hashes before inference, and generated greedily in bfloat16 with batch size eight and at most 448 new tokens. Every model receives 5,184 prompts in the first three conditions and 2,688 in the own-format condition. Two further controls separate the content of the record from its placement: the record format with the successor appended to every line (`At X, x was a and becomes b; next is Y.`) and the model's own step format, both listed after `Execution so far:` in the prompt with the three partial-chain demonstrations rendered the same way (2,688 prompts each per model). First-step transition shares are 0.00–0.01 under both, with final accuracies of 0.74–0.79, 0.94–0.95, and 0.58–0.69 for Qwen2.5-14B, Gemma, and Phi-4 with the appended successor, and 0.82–0.84, 0.97, and 0.75–0.87 in the own step format.

First actions are read by the frozen extractor of Appendix E, which records the node, input value, result, and successor of each asserted step and treats exact restatements of a supplied prefix as history. A step is a transition error when its node or input value differs from the correct next transition, an execution error when node and input are correct but the result or successor is wrong, and indeterminate when the extractor cannot establish the fields; comparison stops at the first such step. Among first-step transition errors after a record, Phi-4 mostly re-enters the last completed node, Qwen2.5-14B mostly jumps to a node off the path, and Gemma does both. Phi-4 writes its own trajectories as itemized lists with the update inside a parenthesis, which the extractor mostly cannot read, so its own-trajectory transition share is not reported, while its supplied-record continuations follow the demonstration syntax and are readable.

Wrong successors in own trajectories. A step that enters the correct node with the correct input and writes the correct result but names a successor other than the one its branch selects is classed above as an execution error of the successor field, since the node and input of that step are correct; the wrong next step it announces is taken at the following step. Such steps are the most frequent first error in Qwen2.5-14B's own trajectories, 165 of 768 (17, 27, 50, and 71 of 192 at depths three to six) against 94 wrong updates, with a per-step hazard of 47/768, 46/666, 39/599, 25/376, 3/207, and 5/83 at steps one to six; Gemma writes 32 (4, 6, 9, and 13) and Phi-4 25 among its 299 readable trajectories. The next step is taken at the named node in 221 of the 222 cases (one trajectory ends there), and 14 of the 222 end with the correct answer. An independent run with its own program generator, prompts, parser, and scoring reads the branch condition, the updated value, and the successor of each step as separate fields on 384 further programs of depth three to six under one wording, for Qwen2.5-14B, Gemma 3 12B, Qwen3-4B with thinking disabled, and Mistral-7B-Instruct-v0.3. A first error with correct condition and value but wrong successor occurs on 94, 13, 52, and 38 of 384 trajectories, and 242, 361, 303, and 29 final answers are correct. On 192 of the programs, demonstrations in which each step writes the selected branch's update and successor before the updated value (`At G, 8 < 9 is true; selected delta: -4; selected successor: P; x becomes 4; next is P.`) change the count of this error from 50, 6, 26, and 20 to 3, 0, 0, and 22 and raise final accuracy from 121, 178, 153, and 11 to 169, 191, 190, and 40, with paired program-bootstrap 95% intervals for the gain of [0.18, 0.32], [0.03, 0.11], [0.14, 0.25], and [0.09, 0.22]; writing the successor before the updated value instead lowers accuracy to 87, 104, 12, and 6, and restating the current value and threshold before each step leaves it at 119, 175, 116, and 12. The same run also continues correct records on the 192-program subset under five conditions sharing the program and remaining computation: a fresh problem from the current state, the record of results, the record plus one sentence naming the next node with demonstrations unchanged, the record under the demonstrations that write the selected branch before each update, and both additions. Correct final answers are 149, 69, 145, 85, and 172 of 192 for Qwen2.5-14B; 186, 112, 185, 113, and 190 for Gemma; 168, 86, 154, 149, and 174 for Qwen3-4B; and 15, 8, 23, 19,

and 57 for Mistral-7B. Wrong first entries under that run's audit, which excludes exact restatements of the history, are 114, 5, 99, and 2 for Qwen2.5-14B, 81, 0, 78, and 0 for Gemma, 139, 11, 44, and 18 for Qwen3-4B, and 155, 0, 152, and 2 for Mistral-7B under the record, with the next node, with the branch demonstrations, and with both.

Table 6: Where the next step goes wrong. Conditional programs of depth three to six, 384 per model, all split points. Own trajectory: execution from the start. Record: the correct steps supplied as results (`x was 10 and becomes 8`). Own-format record: the same steps in the model's step format, naming each successor. Cells give final accuracy under the ordinary and explicit wordings and, for supplied prefixes, the share of first written steps that are transition errors. One step: a single step from a given node and value.

Model	Own trajectory acc.	Record of results acc. / transition		Own-format record acc. / transition		One step acc.
Qwen2.5-14B	0.66 / 0.64	0.42 / 0.50	0.52 / 0.40	0.81 / 0.80	0.00 / 0.00	0.96
Gemma 3 12B	0.94 / 0.95	0.48 / 0.61	0.54 / 0.39	0.96 / 0.96	0.00 / 0.00	0.99
Phi-4 14B	0.64 / 0.76	0.18 / 0.24	0.74 / 0.73	0.89 / 0.89	0.00 / 0.00	0.98

F.2 Arithmetic chains

The arithmetic study uses the answer-balanced natural-format items of depths four to six with their eight demonstrations, rendered as complete chains (`12 - 2 = 10, 10 - 5 = 5, 5 - 4 = 1, 1 + 1 = 2. The result is 2`). Own trajectories start after `A:`; supplied prefixes end after the j-th written step, so the model may write further steps or answer at once; the one-step control asks for the current value and the next operation as a new question. Qwen2.5-3B/7B and Qwen3.5-2B/4B-Base generate greedily with at most 96 new tokens. Written steps are parsed as `a op v = c` and compared in order with the correct chain: a transition error when the input or operation differs, an execution error when the result differs, and a premature answer when the chain ends before the last operation. All outputs, prompts, and classifications are retained in the experiment directory with their hashes.

F.3 Records in other surface forms, trajectories from other models, and arithmetic cues

Three further comparisons use the same 384 programs and the same arithmetic items. *Surface forms.* Six records of the same correct prefix are compared at every split point and under both wordings: the model's own step format with the successor clause deleted (`At X, a < c is true; x becomes b.`), placed either after `Execution so far:` in the prompt or prefilled in the model's own turn; a tabular form without and with the successor (`X: 10 -> 8` and `X: 10 -> 8 -> Y`); and a narrative form without and with the successor (`Step 1 at X changed x from 10 to 8.` and `... and went on to Y.`). Prompt-placed records use the three frozen partial-chain demonstrations with their histories rendered in the same form; prefilled records use the scratch demonstrations. Each form yields 2,688 prompts per model. *Other models' trajectories.* From the own-trajectory outputs under the ordinary wording, we keep the programs on which Qwen2.5-14B (244 programs) or Gemma (358 programs) wrote every step correctly, cut the response verbatim at the end of the t-th asserted step as delimited by the frozen extractor, and prefill it in the other two models' turns after the scratch demonstrations; no wording or formatting of the source text is changed. Qwen2.5-14B continues Gemma's trajectories with final accuracy 0.826 and one wrong first step in 1,238 prompts, Gemma continues Qwen's with 0.971 and two in 788, and Phi-4 continues Qwen's and Gemma's with 0.933 and 0.893 and none. *Arithmetic cues.* Under the immediate-answer readout, four conditions share the eight demonstrations rendered in the matching form: the record alone (`... = 5. The result is`), the record followed by `Next: subtract 4.`, the record followed by `Completed 2 steps.`, and the value-plus-remainder restatement; accuracy is the closed score over 0–20 with a four-token greedy readout, on all twelve depth–split cells of the S210 items for Qwen2.5-3B/7B and Qwen3.5-2B/4B-Base. All prompts are frozen with hashes before inference and all outputs are retained.

Interpreter check and uncertainty. Running the independent interpreter of Appendix E over every record prompt of the record condition recovers the next node, the current value, and the answer from the same text on 2,688 of 2,688 prompts. Wilson 95% intervals for the proportions in Tables 6 and 7 have half-widths of at most 0.02 at 2,688 prompts per cell and at most 0.04 for the cross-model cells; every contrast between a record without the successor and the same record with it exceeds these intervals by an order of magnitude. The three partial-chain demonstrations shown with every record continue from the correct next node, so the model has seen how the successor follows from a record and still fails on 0.39–0.74 of prompts.

Table 7: What the record states about the next step decides; surface form does not. First-step transition share and, in parentheses, final accuracy for records of the same correct steps in different surface forms, with and without the successor; wordings pooled, 2,688 prompts per cell. The record of results in Table 6 is the reference without a successor. The own-format rows delete only the successor clause and still state each branch outcome, an intermediate amount of information about the next step.

Form of the record	Says what comes next	Qwen2.5-14B	Gemma 3 12B	Phi-4 14B
Own step format, in the prompt	no	0.26 (0.56)	0.24 (0.64)	0.81 (0.21)
Own step format, in the model's turn	no	0.22 (0.42)	0.17 (0.41)	0.19 (0.35)
Table of value changes	no	0.46 (0.47)	0.54 (0.48)	0.89 (0.14)
Table of value changes	yes	0.01 (0.80)	0.00 (0.95)	0.01 (0.95)
Narrative sentence per step	no	0.46 (0.45)	0.46 (0.56)	0.89 (0.14)
Narrative sentence per step	yes	0.01 (0.80)	0.00 (0.95)	0.00 (0.92)

G Context and implied-boundary diagnostics

G.1 Value adoption and question context

Table 8 groups the symbolic controls around two comparisons. With the value and remainder fixed, changing the history has a relatively small effect; replacing the value instead redirects execution toward the replacement. With the local expression fixed, retaining a question that describes completed operations incurs an additional cost. Re-asking the remainder recovers much of that cost, while repeating the original question does little in Ouro. The cross-family comparison appears in Figure 10.

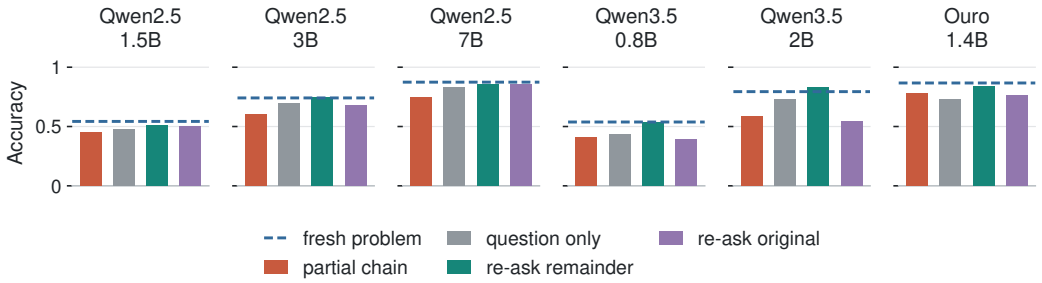

Figure 10: Context affects execution even with an explicit remainder. Accuracy on the same four-operation remainder under different contexts. Re-asking the remainder helps across families. Re-asking the original question also helps Qwen2.5, but has little benefit in Ouro and the smaller Qwen3.5 models. Standard-model means use their common paired item sets; Ouro uses its reference context comparison.

In the replacement-value comparisons, the original answer is produced on at most 0.004 of items, and merely copying the replacement accounts for none of the outputs. At three remaining operations, the tested histories meet the original equivalence rule against the fresh reference. With four operations left, changing the original question's operands redirects only 0.04–0.05 of answers toward the

Table 8: Symbolic controls in Ouro-1.4B. Replacement-value adherence is measured from pass two onward; other rows give fourth-pass accuracy with four operations remaining. History, question, and re-asking comparisons use their original paired sets. The final block varies how much completed work the question describes.

Comparison	Rate
Executing from a replacement value	
Depth four	0.996–0.998
Depth five	0.944–0.952
Depth six	0.93–0.96
Different histories, fixed value and remainder	
Original derivation	0.776
Different derivation, same initial value	0.756
Different initial value	0.746
Unrelated expression / contradictory derivation	0.729 / 0.726
Original question and re-asking	
Fresh problem	0.867
Original question + local expression, no derivation	0.731
Partial chain + local expression	0.776
Partial chain, remainder re-asked	0.838
Partial chain, original question re-asked	0.767
Question only, remainder re-asked	0.845
Fresh problem, question repeated	0.867
Question describes 1 / 2 / 3 completed operations	
With evaluated chain	0.740 / 0.771 / 0.790
Without evaluated chain	0.790 / 0.731 / 0.750
Fresh reference for this comparison	0.869

corresponding altered target. The surrounding question affects execution even though its operands rarely replace the local expression's inputs.

Text deletion and attention masking provide different controls. Deletion removes information while leaving a contiguous prompt. Masking keeps the tokens and their positions but blocks selected attention paths. The context comparisons use text edits; the matched-supervision experiment uses masking to test dependence on information in the original question (Appendix I).

In Ouro, a question describing completed work incurs a step-like penalty that changes little as further completed operations are added. Altering its operands rarely redirects the answer toward those alterations. Qwen2.5 benefits from deleting the prefix, while equally long filler retains a similar cost, consistent with recency or length effects. In natural-format Ouro, even length-matched filler can outperform a correct partial chain at depths four and five: an advanced written result can be harder to use than the original starting point.

Partial-chain answers include neighboring implied boundaries and, in Qwen3.5-2B, applying only the final operation. Unmatched answers remain common. A comparison against contiguous windows finds repetition of consumed operations above a uniform-error reference, unlike fresh-problem errors. Together with the changes under boundary supervision, this favors learned alignment to the number of completed steps as an interpretation. The classification measures which execution would produce the answer; it does not read a hidden counter.

G.2 Classifying implied boundaries

For every answer, we execute all suffixes from the supplied value. A unique match determines $\hat{j}$; multiple matches and outputs matching no suffix are retained separately in the saved counts and combined in the plotted "?" column. Rows normalize over all items at a true boundary. A diagonal entry therefore measures a uniquely identified intended suffix, while answer accuracy can include ambiguous matches. Figure 11 shows depth five, and Figure 12 extends the comparison to depth six.

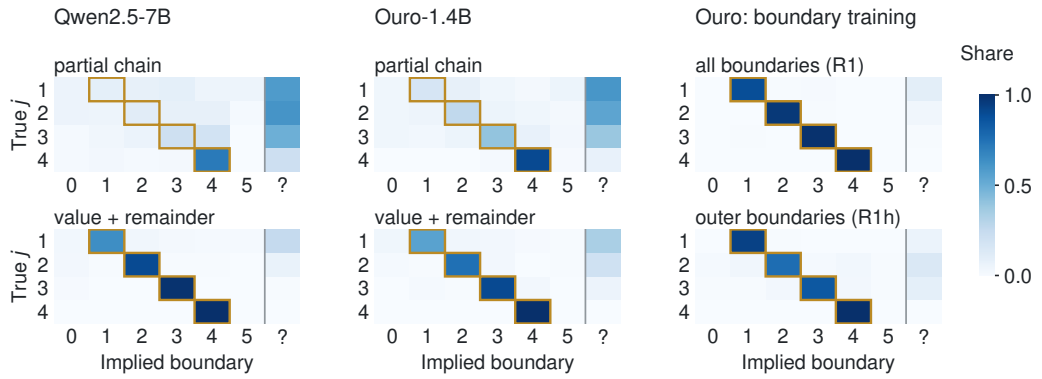

Figure 11: True and implied continuation boundaries at depth five. Rows give the actual split j; columns give the suffix uniquely producing the answer from s_j. Outlines mark intended boundaries; "?" collects ambiguous or unmatched outputs. Remainder descriptions and R1/R1h supervision change these alignments.

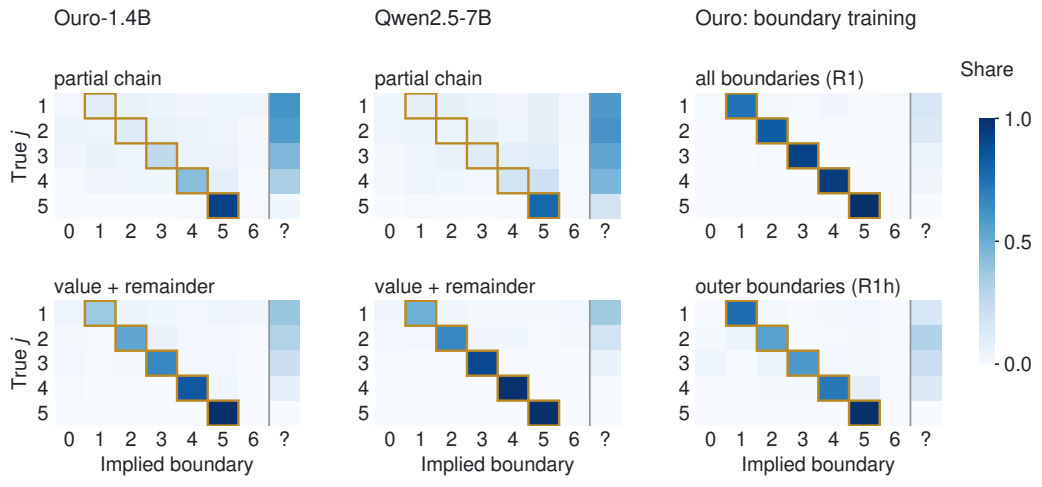

Figure 12: Implied boundaries at depth six. Rows give the true split; columns give the uniquely matching suffix. Outlined entries mark the intended boundary, and "?" combines ambiguous and unmatched answers. The same classification as in Figure 11 is used at a longer depth.

H Residual-state interventions

H.1 Donor positions, recurrence, and uncertainty

The donor records the pre-normalization residual stream after a selected layer and recurrent pass. At the same point in a recomputed receiver run, the selected positions are overwritten, and the receiver finishes its remaining layers and passes. Other positions retain the receiver's states and key–value cache. Remainder-position imports align the identical final expressions by their offset from the prompt's end. The different-item answer-position comparisons replace only the final prompt position; the matched-supervision comparison below imports a span covering the answer.

The four-operation comparison uses 398 items and a fresh-minus-partial accuracy gap of 0.095; the five-operation comparison uses 291 items and a gap of 0.172. Recovery divides the paired improvement by the original gap as in Equation 1. The position-matched donor is recomputed at the receiver's absolute positions, controlling rotary position phase. Figure 13 retains the signed effects, including cells below zero, and uses the original paired bootstrap estimates. Every intervention is read at the receiver's fourth pass.

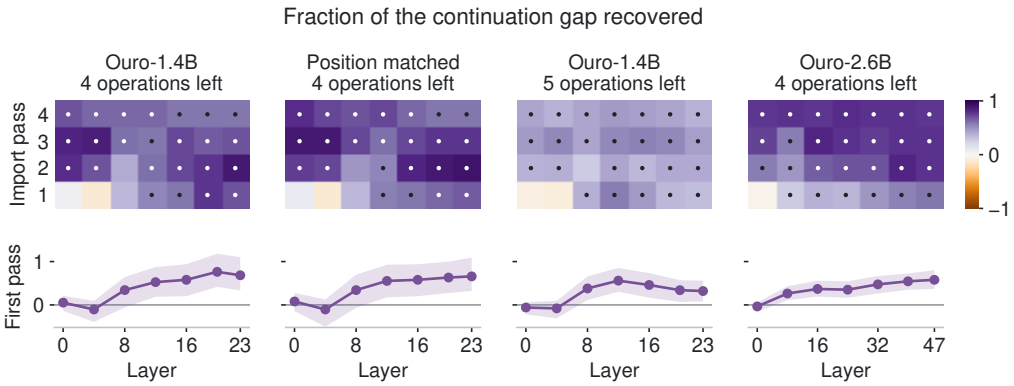

Figure 13: Importing a fresh run's remainder states repairs continuation. Color gives the fraction of the gap recovered at pass four; negative values indicate further loss. Panels compare remainder length, position matching, and model size. Dots mark paired intervals wholly above zero. Profiles show first-pass estimates and uncertainty (Appendix H).

H.2 What transfers across problems?

In the four-operation comparison, different-item remainder imports during the first pass yield the donor's answer on 0.63–0.74 of items. These positions carry the expression's content as well as a state useful for executing it. Figure 14 examines answer-position donors at equal or neighboring split points. Categories distinguish the receiver's answer, the donor's answer, executing from the receiver value at the donor boundary, and other outputs. Later imports increasingly transmit the donor's answer. The hybrid boundary target remains rare, distinguishing these imports from a reusable, item-independent boundary signal.

H.3 Same-item imports between supervision regimes

The broad and restricted arms receive identical prompts. The imported span contains the final prompt position and every teacher-forced gold-answer position, including the newline. Both runs receive the same token sequence; the intervention replaces residual states across this span, and the receiver finishes the remaining layers and passes. Strict token accuracy requires each gold answer token and the newline to be the argmax at its preceding position. Figure 15 shows all three seeds and a reverse-direction comparison. Recovery begins within the first pass and approaches donor performance by the next pass; the reverse effect is weaker. This establishes the usability of the donor's current-answer computation across the span, without localizing it to a single position.

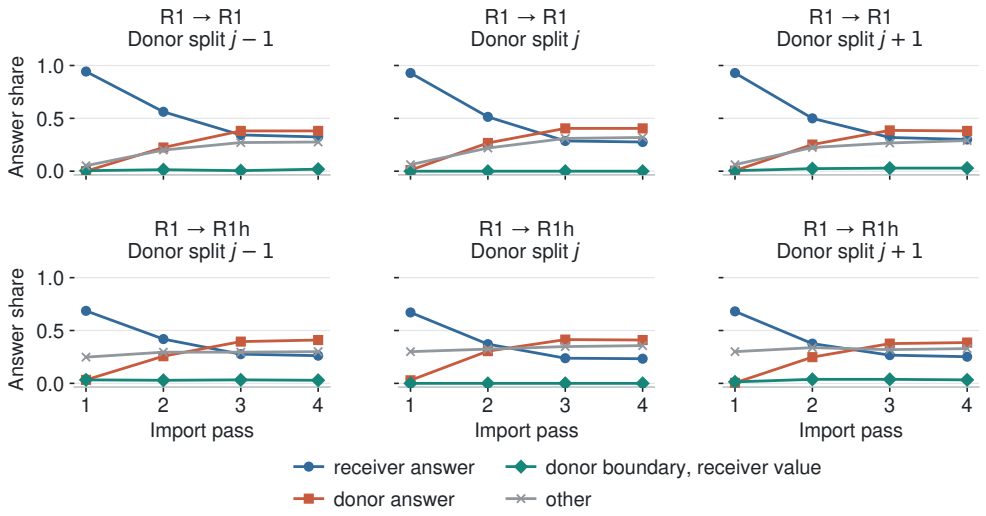

Figure 14: Answer-position imports across problems. Donors come from a different item at the last layer of the indicated pass. Columns vary the donor split relative to the receiver; rows use R1 or R1h receivers with an R1 donor. Curves report answer categories at the receiver's fourth-pass readout, separating donor content from a hybrid use of the donor boundary.

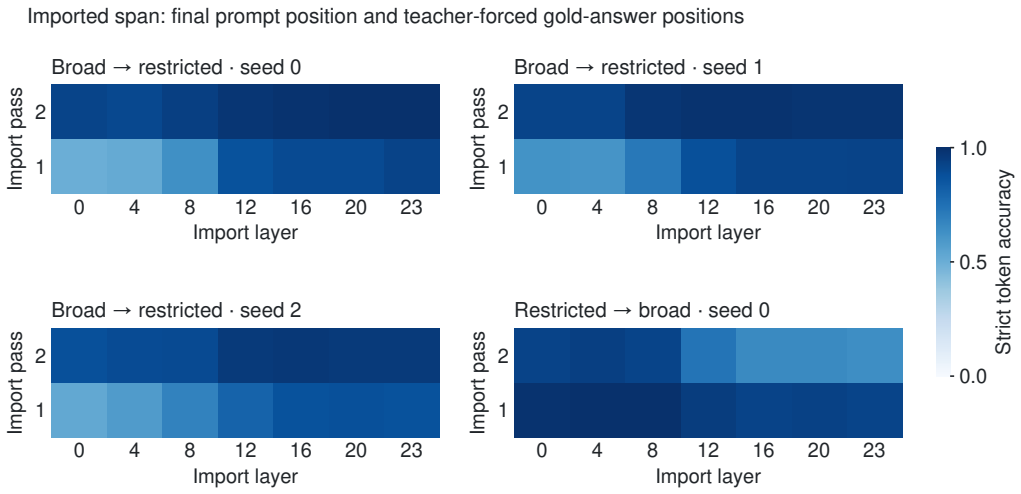

Figure 15: Same-item answer-span imports at a withheld entry. Strict token accuracy after importing the final prompt position and teacher-forced answer positions at the indicated layer and pass, followed by the receiver's fourth-pass readout. The first three panels show broad-to-restricted imports by seed; the last reverses the direction.

Table 9: Interior-boundary accuracy across supervision regimes. Withheld-regime entries span three seeds. R1 values are the corresponding reference accuracies. W regimes omit only one cell.

Withheld regime	(d, j)	R1	Withheld
R1h	$(4, 2)$	0.99	0.88–0.93
	$(5, 2)$	0.98	0.70–0.76
	$(5, 3)$	1.00	0.69–0.85
	$(6, 2)$	0.85	0.50–0.56
	$(6, 3)$	0.90	0.50–0.60
	$(6, 4)$	0.99	0.65–0.72
W52	$(5, 2)$	0.98	0.87–0.94
	$(6, 2)$	0.85	0.54–0.76
W53	$(5, 3)$	1.00	0.96–0.98

I Boundary supervision

I.1 Coverage and optimization

R1 samples all $j \in \{0, \ldots, d\}$; R1h samples only $j \in \{0, 1, d-1, d\}$. W52, W42, and W53 each omit the indicated (d, j) cell while retaining its neighbors. Training depths are two to five, so depth six measures transfer beyond the trained lengths. Figure 16 displays the supervised cells and evaluation accuracy. Table 9 summarizes the interior-boundary seed ranges, separating omission of a boundary type from omission of a single cell.

All adapters use Ouro-1.4B with LoRA rank 16, $\alpha = 32$, and dropout 0.05 on query, key, value, output, gate, up, and down projections. Embeddings, output heads, normalization parameters, and the exit gate are frozen. AdamW runs for 1,000 steps with effective batch size 16, initial learning rate 2×10^{-4}, 50 warm-up steps, and cosine decay to 2×10^{-5}. The loss averages four recurrent heads. R1, R1h, and W regimes use 8,000 chains and supervise all tokens after `A:`; matched arms use 8,000 paired rows and supervise the final answer. Each design has three seeds and a common evaluation readout.

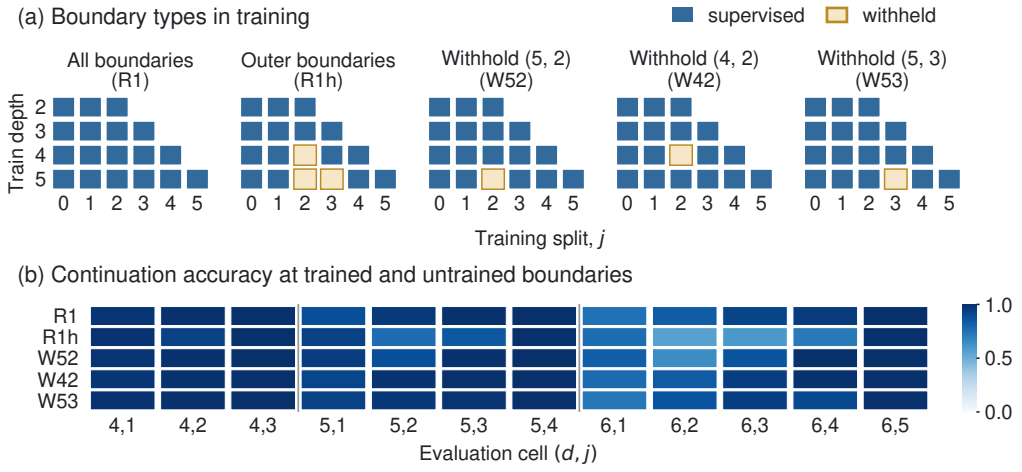

Figure 16: Boundary coverage shapes continuation. (a) Supervised and withheld entries. (b) Accuracy for R1/R1h seed one and W seed zero; thin subdivisions preserve saved evaluation variants. Depth six exceeds the training range. Table 9 gives seed ranges.

At boundary types shared by R1 and R1h, performance remains similar: at depth six with one written step, R1h scores 0.76–0.84 against 0.77 for R1, and with one operation left both reach 1.00. Withholding $(4, 2)$ alone also preserves performance. Under omission of all interior boundaries, common errors execute only the final operation or restart the full sequence from the written value. The share repeating completed operations is 0.5–0.7, above the 0.3–0.4 chance reference.

Table 10: Matched supervision arms and context controls. Mean strict token accuracy across three seeds, with 147 paired items per seed; every answer token and its newline must be the greedy argmax. The withheld entry, supported entry, and fresh problem share the same three-operation remainder.

Condition	Restricted	Broad
Withheld entry	0.556	0.984
Supported entry	0.991	0.991
Fresh problem	0.993	1.000
Withheld entry, remainder re-asked	0.989	0.998
Question only, no chain	0.841	0.859
Fresh question + partial chain	0.018	0.020
Withheld entry, question masked	0.052	0.034
Withheld entry, chain masked	0.599	0.619
Withheld entry, demonstrations masked	0.000	0.000

I.2 Matched entry conditions

The broad arm covers $(3,1),(3,2),(4,1),(4,3),(5,2),(5,3)$; the restricted arm covers $(3,2),(4,1),(5,3)$. Paired rows share the remaining computation, final-answer target, and demonstrations, while the marginal distributions of total depth, completed steps, and remaining length are matched. The arms differ in which depth–boundary pairs receive target supervision; the withheld pairs still occur in their shared demonstrations. Changing the history to obtain a different entry changes the full question, so this is a match of the remaining computation rather than identical training prompts. Fresh and zero-step rows and the optimization budget are shared. Table 10 compares the withheld entry with a supported entry into the same remainder and its fresh formulation. It also retains question and attention controls. A separate greedy-integer readout generates at most three tokens and parses the answer without requiring the newline. Of its 199 restricted-arm errors, 73% match execution of a wrong contiguous window from the correct current value, most often the alignment supported at that depth.

I.3 Matched training in a conventional Transformer

Qwen2.5-3B uses the same 8,000 paired training rows per arm: 4,800 partial rows, 1,600 zero-step rows, and 1,600 complete traces. The three paired seeds share initialization and example order. LoRA uses rank 16, $\alpha = 32$, dropout 0.05, and the same seven projection types as above. AdamW takes 1,000 updates with microbatch eight and accumulation two, learning rate 2×10^{-4}, 50 warm-up updates, and cosine decay to 2×10^{-5}. Each example has equal weight after averaging loss over its final answer and newline; the prefix has zero loss. The sole confirmation endpoint is the final update.

For each of 147 three-operation remainders, H is $(d, j) = (5, 2)$, S is $(4, 1)$, and fresh execution is $(3, 0)$. The broad arm receives H and S as supervised targets; the restricted arm receives S. Both have the same demonstrations, including H. Figure 17 retains each seed. All fresh and S controls exceed 0.95, while the restricted H scores range from 0.306 to 0.939. Strict digits-plus-newline accuracy and greedy parsed-integer accuracy agree on the primary confirmation outputs. Intervals in the numerical artifact resample remainder families jointly across arms and seeds; they are conditional on these fitted runs and do not estimate between-seed uncertainty.

Figure 18 shows every reminder at H. The pointer states the completed count and next operation index without repeating its content. Value + remainder gives the current result and an inline list of operations. These sentence-form reminders can perturb even the broad model's otherwise accurate continuation. A supplementary comparison, fixed after this primary evaluation, instead uses the exact earlier question–answer re-asking protocol: after the original question and trace, it presents `Q: Start with` the current result and remaining operations, followed by `A: The result is`. The same six adapters and 147 remainders are retained, giving 882 additional outputs. Accuracy returns close to ceiling in both arms. This contrast changes task framing along with the explicit remainder, and shows that the two forms of restatement have different effects.

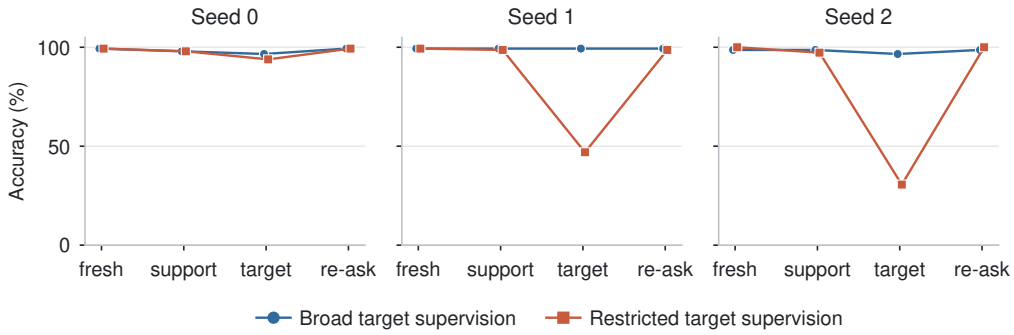

Figure 17: Matched training support affects Qwen2.5-3B continuation. Each panel shows a pair of training runs using the same seed on the same 147 three-operation remainders. The target entry receives target supervision only in the broad arm; it occurs in shared demonstrations for both. Fresh and supported entries control execution ability. Re-asking the remainder uses the original question–answer format and is a separately frozen supplementary comparison. Points retain the large variation between restricted runs.

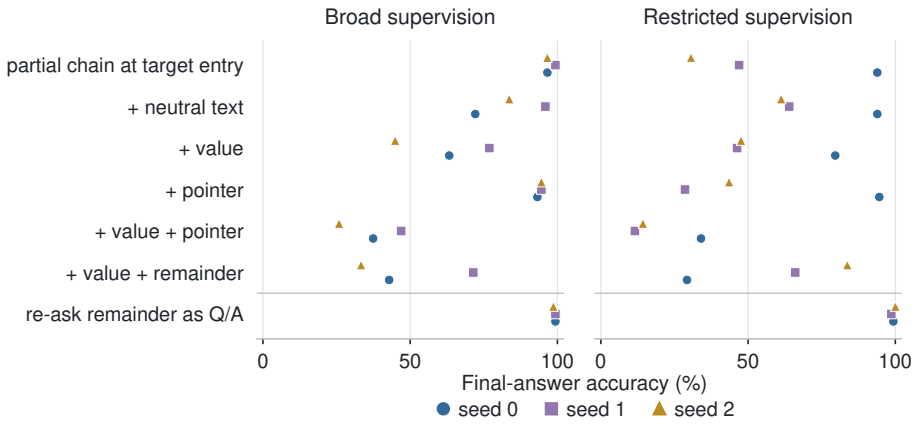

Figure 18: The form of a reminder matters for the trained interface. Every point is one fitted seed evaluated at H on the same 147 remainders. All sentence-form reminders retain the original question and correct trace. The separately frozen Q/A re-ask presents the identical remainder as a new question after that history. All original cue results remain visible, including decreases under broad supervision.

Table 11: Continuation with an internal multiplication. Natural-format accuracy with two operations remaining and 210 paired items.

Model	fresh problem	partial chain	chain + value + remainder
Ouro-1.4B	0.990	0.690	0.986
Qwen2.5-7B	1.000	0.733	0.990

J Operators and discrete state spaces

J.1 An internal multiplication

Multiplication chains have depths two to five, with one internal multiplication by two or three. Table 11 gives the natural-format comparison with two operations left; half of these remainders contain the multiplication. With three operations left, all contain the multiplication and fresh accuracy is 0.29 in Ouro-1.4B and 0.42 in Qwen2.5-7B. The shorter execution horizon is visible in the symbolic sweep (Figure 20) and is separated from the cost of entering through a trace.

J.2 Heading and card programs

Heading states are north, east, south, and west; operations turn left, right, or around. Three-card states are orderings of three distinct cards drawn from A–H. Operations swap the first pair, swap the last pair, move the last card to the front, or move the first card to the end. Sampling excludes chains that revisit an earlier card ordering. Heading depths are three to six and card depths three to five, with 210 items per cell and eight same-task demonstrations whose split points vary.

For example, the heading question "You face north. Turn left, then turn around, then turn right. Which way do you face at the end?" has the trace "west, east, south." A partial chain stops at the chosen state. A fresh problem starts from that state and lists only the remaining actions; the other conditions repeat the state, remaining instructions, or both. Greedy outputs are matched exactly to a direction or card ordering. Uniform guessing gives 0.25 for heading and approximately 0.17 for cards.

Figure 19 shows both the fresh-solving horizon and all one-operation continuation comparisons. In the one-operation card comparisons, outputs consistent with repeating completed operations are more common than outputs consistent with skipping ahead. Heading errors are more dispersed. The small state spaces allow accidental matches to alternative programs, so this classification supplies a directional description rather than a unique inferred boundary.

K Python execution with loops and conditional branches

K.1 Programs and matched execution states

The task uses an eleven-line Python template with two integer variables and a loop index. Conditional updates change one variable, a merge statement updates the other, and the index advances. The loop revisits source lines. We pause at one of four positions within an iteration: before the condition, the selected branch update, the merge update, or the index update. The 128 confirmation programs contain 32 examples of each pause type; 16 pilot programs are disjoint. The following is one saved confirmation example.

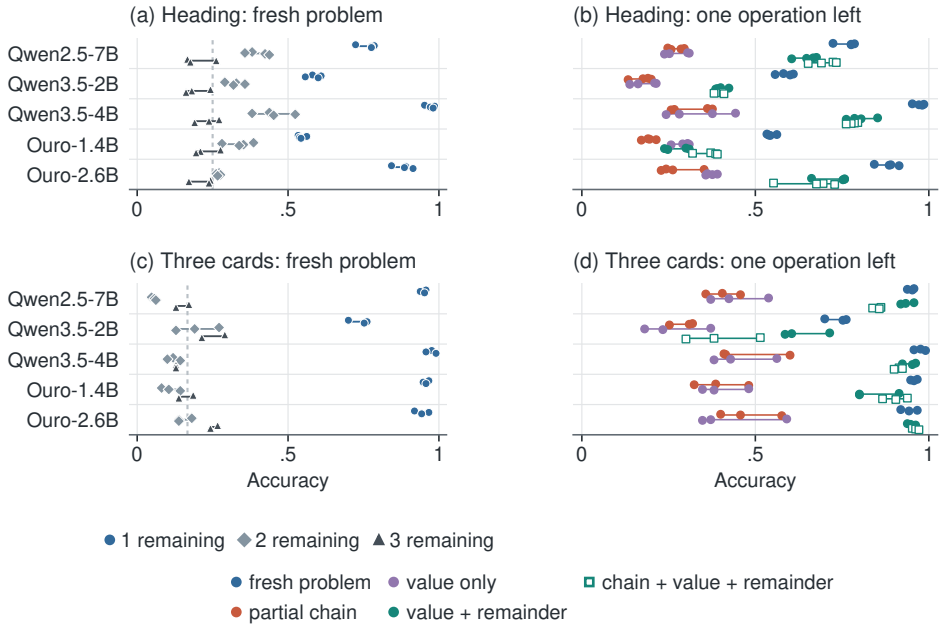

Figure 19: Execution horizon and continuation in discrete programs. Left: fresh accuracy at each tested remaining length; dashed lines mark uniform guessing. Right: all five continuation conditions with one operation left. Markers are saved model–depth–split scores, and short lines span their observed range, not a confidence interval. All cells contain 210 paired items and use greedy readout.

```
x = 1                  # L1
y = 6                  # L2
i = 0                  # L3
while i < 4:           # L4
    if (x+i)%4 == 3:   # L5
        y = y + 1      # L6
    else:              # L7
        y = y - 1      # L8
    x = x + y          # L9
    i = i + 1          # L10
answer = x             # L11
```

At the selected pause, the frame is $x = 12, y = 5, i = 2$, and the next line is L9. The remaining execution produces 21. The same program also reaches $x = 12$, $y = 6$, $i = 2$ before both L5 and L8, so the frame alone does not uniquely identify the next instruction. Native Python tracing and an independent transition interpreter verify the complete execution and every candidate suffix. Re-entry tests retain the original code and load the specified frame and line. The paired prompts therefore have the same remaining execution, including branch decisions and repeated line visits.

K.2 Prompt controls and final-answer evaluation

Every condition contains the identical full source, including its initial assignments. Fresh gives the current frame and next line without history. The partial condition supplies the completed execution; its reminders add the frame, next line, or both. Event and statement formats render the same history, with the fresh condition shared. Each item has two index updates left; the number of remaining executed lines depends on the pause position. The loop index itself carries progress information, so these are tests of program resumption rather than a pure separation of value and position.

The three models use their released chat templates and greedy generation. A fresh-only pilot selects the common 1,024-token allowance. The primary response format requires only a marked final answer and permits free reasoning. Table 12 retains all programs: five unparsed responses and one capped response count as incorrect. Because the complete source and initial state are visible, a correct final answer can also result from recomputing the program; this endpoint measures performance under each continuation interface, not fidelity to a particular internal execution path.

Trace	Model	fresh	partial	+ frame	+ next line	+ both
Events	Qwen2.5-14B	99	21	19	28	33
Events	Gemma 3 12B	124	77	59	85	68
Events	Phi-4	112	85	73	91	109
Statements	Qwen2.5-14B	99	61	58	60	53
Statements	Gemma 3 12B	124	80	70	79	77
Statements	Phi-4	112	87	73	91	111

Table 12: Continuation of Python programs with loops and conditional branches. Each entry counts correct final answers out of 128 fixed programs. All prompts contain the same full source. Fresh supplies a variable frame and the next source line; the other conditions include the completed execution history, optionally repeating the frame, the next line, or both. Events and statements render the same history in two ways, sharing the fresh condition. All responses are freely generated with a final-answer marker.

L Written steps and recurrent budgets

L.1 Supplied intermediate values

Figure 20 summarizes the complete supplied-state sweep. The number of remaining operations explains much of the variation: one- and two-operation remainders are nearly exact from the second pass onward, while longer ones retain a cost relative to fresh chains of the same length. At depth four, an expression with no supplied intermediate value changes little after the trained recurrent range. Extra passes do not reproduce the improvement from writing a value within the chain.

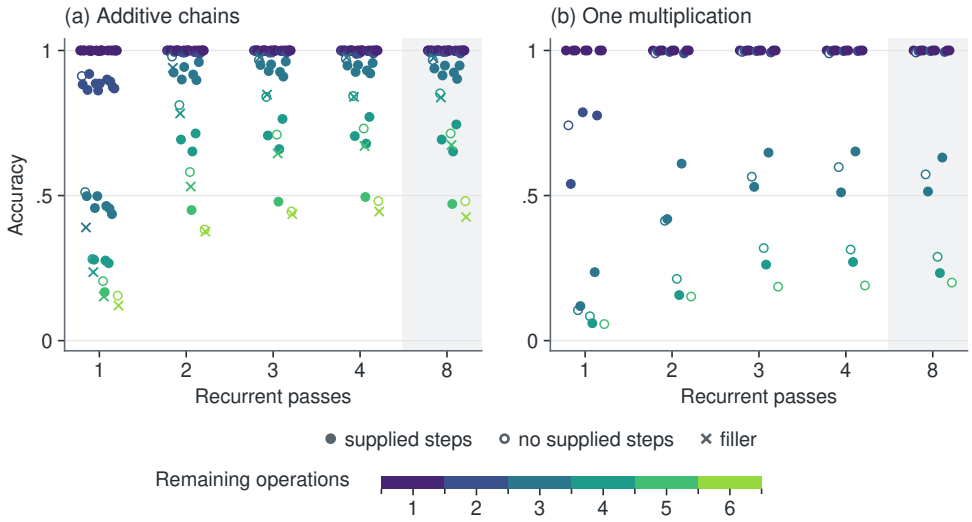

Figure 20: Supplied-state accuracy across recurrent budgets in Ouro-1.4B. Each point is an evaluated configuration: 48 additive and 17 one-multiplication configurations at five budgets. Color gives the length after the last supplied value; open markers have no supplied step and crosses add filler. Horizontal jitter separates configurations without changing accuracy. The shaded pass-eight region exceeds the four-pass training range.

L.2 Generated intermediate values and computation cost

The end-to-end sweep lets Ouro write its own intermediate states under the demonstrated pattern before answering. Both the written states and final answer are generated greedily; accuracy uses the parsed final integer. Figure 21 reports every evaluated configuration and its observed cost–accuracy frontier. At depth four, writing every second step with two passes for both writing and answering

reaches 0.962 accuracy at 2.93 thousand block applications, compared with 1.000 at 3.50 thousand when every step is written. At depth six, writing each step at that budget reaches 0.995. These results include errors in producing intermediate values as well as errors in continuing from them.

Costs count block applications with the demonstration prefix cached. If the writing and answering budgets differ, the evaluator recomputes every position at the answering budget, and this cost is included. The frontier describes the evaluated implementation under that convention. A product reference multiplies fresh-chain accuracies for the generated segments and final remainder at their respective budgets; parsed segment lengths determine the per-generation reference, and malformed generations count as incorrect.

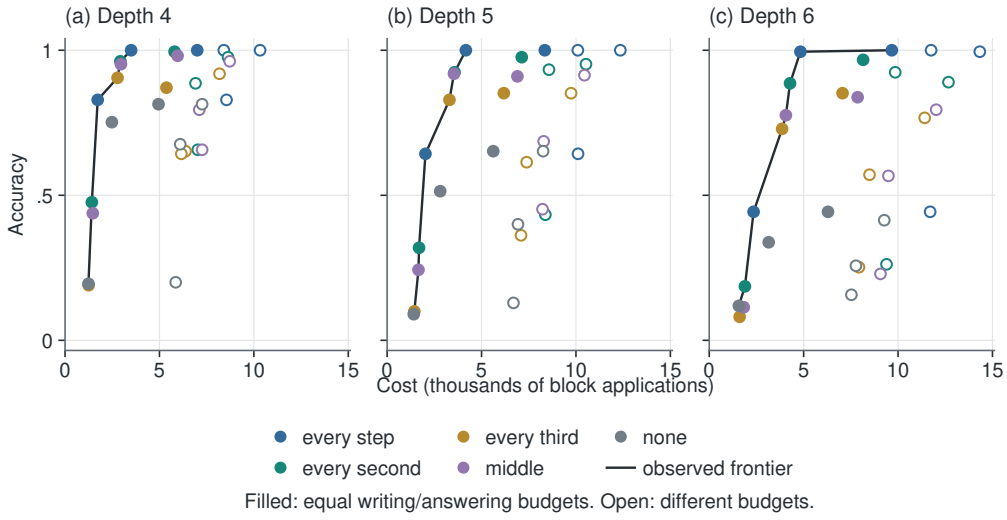

Figure 21: Cost and accuracy when the model writes its own intermediate states. All 90 configurations are shown, comprising five writing patterns, six writing/answering budget pairs, and three depths, with 210 items per configuration. Lines connect observed nondominated points within a depth. Filled markers use equal writing and answering budgets; open markers use different budgets. Costs include the evaluator's cache recomputation when the budgets differ.