OUT OF SCOPE, STILL ACTIVE: DO LANGUAGE MODELS RESPECT REASONING SCOPE?

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models increasingly solve complex problems by exploring hypotheses, revising intermediate conclusions, and moving between local stages of reasoning. Such reasoning is hierarchical and requires *selective carry-over*: after a local stage ends, conclusions that remain supported should carry forward, while those supported only locally should stop guiding inference. We introduce *reasoning with scope* (RWS), an analytical framework based on scoped computation in programs, to study how LLMs carry information across reasoning stages. Across controlled settings and natural reasoning traces, information from a completed scope can continue to influence later reasoning, while still-supported conclusions can be weakened. Crossed tests show that models respond more strongly to where a conclusion was derived than to whether it remains supported. This pattern changes substantially when the same reasoning is expressed more rigorously as an executable program: in natural-language reasoning, models often treat supported and unsupported conclusions similarly, whereas under executable scope their behavior can differ by nearly 90 percentage points. Reliable reasoning therefore requires tracking not only what has been inferred, but also which inferences remain justified as reasoning moves across stages.

1 INTRODUCTION

Recent advances in large language models (LLMs) have enabled increasingly complex tasks, including multi-step reasoning, software engineering, tool use, and interactive decision making (Wei et al., 2022; Yao et al., 2023b; Zhou et al., 2024; Yang et al., 2024b). Work on test-time scaling and reasoning models further shows that allocating more inference computation can substantially improve performance on difficult problems (Snell et al., 2025; Guo et al., 2025; Muennighoff et al., 2025). Together, these advances are pushing LLMs from one-shot generation toward sustained problem solving on increasingly complex tasks.

Complex reasoning rarely follows a single linear path. Models decompose tasks, explore alternative hypotheses, backtrack, and revisit earlier ideas, creating nested local stages under different assumptions, evidence, and constraints (Yao et al., 2023a; Saparov & He, 2023; Lee et al., 2026). Our trace analysis shows that such changes of direction are common and that abandoned ideas can remain influential after reasoning moves on (Appendix A.1). This creates a selective carry-over problem: useful results should remain available, while conclusions supported only within a completed stage should stop guiding later inference. The same qualitative pattern appears in a few natural reasoning traces from distinct AIME problems (Appendix A.2).

This question matters beyond diagnosing a reasoning quirk. Modern process supervision and reasoning-oriented reinforcement learning optimize intermediate computation, so their training signals should capture not only whether a step is locally correct, but whether it remains justified after its assumptions expire (Lightman et al., 2024; Gao et al., 2024; Guo et al., 2025). The same issue arises in long-horizon agents that interleave reasoning, actions, observations, and revisions: reliable systems must retire stale local conclusions without discarding results that remain valid (Yao et al., 2023b; Yang et al., 2024b; Huang et al., 2026). Together, these observations expose a shared design problem for training and agent memory: managing the validity lifecycle of intermediate in-

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.

formation. Recent work has therefore analyzed several complementary aspects of complex LLM inference (Table 1). Reasoning-dynamics studies characterize branching, backtracking, and self-correction (Wang et al., 2025; Lee et al., 2026), but do not determine what information from a completed local stage should remain available afterward. Faithfulness and grounding studies identify which earlier information influences or supports later conclusions (Lanham et al., 2023; Bogdan et al., 2025; Ling et al., 2023; Nakamura, 2026), but do not analyze how these dependencies should be handled when local conditions end. Belief-updating and recovery studies examine whether models can revise earlier conclusions or escape unhelpful reasoning (Wilie et al., 2024; Yang et al., 2025b), but do not distinguish what from the previous stage should carry forward from what should remain local. State-lifecycle and rollback work studies when cached state can be retired or restored across agent transitions (Huang et al., 2026; Zhang & Yang, 2026), but not which logically supported conclusions should survive a reasoning-stage boundary. We focus on these missing aspects of LLM inference analysis. Table 1 uses four broad lenses: *hierarchy* asks whether reasoning is organized into nested stages; *dependency*, which earlier information supports a conclusion; *transition*, what happens when reasoning changes stage; and *selective carry-over*, which information should remain active afterward. The entries summarize each line's primary emphasis rather than exhaustively classifying every paper.

Table 1: **Qualitative positioning of representative research lines.** ✓: primary; △: partial; ×: not central.

Research line	Hierarchy	Dependency	Transition	Selective carry-over
Reasoning dynamics	△	△	✓	×
Faithfulness & causal analysis	×	✓	×	×
Belief updating & recovery	×	△	✓	△
Logical grounding	×	✓	×	×
State lifecycle & rollback	×	△	✓	△
Ours: scope analysis	✓	✓	✓	✓

To analyze this question cleanly and distinguish scope analysis from adjacent lines of work (Table 1), we begin from two common properties of complex LLM inference. First, inference is *hierarchical*: a complex task typically contains nested subproblems, each with its own assumptions, constraints, and intermediate results. Second, inference repeatedly moves between these local stages, and each transition requires *selective carry-over*: useful results or lessons should remain available, while conclusions supported only by local conditions should no longer guide later inference. Together, these properties imply that intermediate information can have different validity after the stage in which it was produced.

These two properties closely parallel how programs structure computation. Programs are hierarchical: functions and nested blocks introduce local variables and conditions within a larger computation. At each boundary, only selected information is exposed through return values or updates to outer state, while certain details remain local. This parallel motivates *scope* as an abstraction for LLM reasoning: a reasoning stage has local context, but only information that remains justified should carry forward after the stage ends. We use this analogy to define *reasoning with scope* (RWS), our framework for analyzing how models manage the validity of intermediate information across stage boundaries.

Analyzing RWS in natural language raises two challenges. First, a reasoning trace rarely states where a local stage begins or ends, or which conclusions rely on that stage's assumptions. The grouping of steps in the text may also differ from their logical dependencies (Bogdan et al., 2025; Lee et al., 2026). Second, scope and position change together. A conclusion derived inside a stage also appears in a different part of the text from one derived outside it. If the two are treated differently after the stage ends, we must determine whether the model tracked their remaining support or merely reacted to their position; premise order is known to affect LLM reasoning (Chen et al., 2024).

To make scope measurable, we introduce a *scope-transition diagnostic* that asks a direct question: after a stage ends, does the model retain conclusions that remain supported and stop using those whose support has ended? We design matched examples that separate this behavior from where a conclusion appeared in the text, and test the same effect in natural reasoning traces using state interventions. This lets us attribute the behavior to scope handling rather than to a general tendency to reuse earlier context. Our main contributions are:

- **We introduce reasoning with scope (RWS) as an analytical framework for complex LLM inference.** RWS studies both how reasoning is organized into scopes—with boundaries, hierarchy, and dependencies—and how information should selectively carry across

scope transitions. Our scope-transition diagnostic makes the second question measurable (Sections 2–4.1).

- **We uncover a systematic failure at reasoning-stage transitions.** Conclusions whose support has ended can remain influential, while independently supported conclusions can be weakened or fail to carry forward. Across matched comparisons, models respond more to where information appears in the reasoning trace than to whether its supporting assumptions remain active (Tables 2–5).
- **We identify a sharp gap between visible scope markers and executable representations.** Adding explicit scope markers to natural language provides little benefit, whereas expressing the same dependency structure as executable programs produces substantially stronger sensitivity to which conclusions remain supported. This shows that visibly marking a boundary is insufficient, while leaving open which properties of executable representations produce the stronger sensitivity (Tables 6 and 7).

2 Scope in Language Model Reasoning

Prior work has examined several aspects of language-model reasoning. ReasoningFlow classifies reasoning steps and their relations (Lee et al., 2026), while Thought Anchors measures how earlier steps influence later ones (Bogdan et al., 2025). As LLM reasoning grows more complex and hierarchical, step-level analysis alone cannot explain how information should be handled across reasoning stages. This calls for modeling reasoning at the level of hierarchical scope: each stage has a local context, and its boundary determines what can carry forward. We formalize these two properties as hierarchy and selective carry-over.

2.1 Hierarchy: Local Context and Derivation Support

Complex inference is hierarchical. A model can enter a subproblem under a temporary assumption, reason within it, and return to an enclosing problem. The subproblem inherits the enclosing context and adds its own assumptions. Its conclusions can therefore draw on different sources of support. Suppose we temporarily assume it will rain while planning an event. We may conclude that the event needs an indoor venue. In the same discussion, we may calculate from the guest list that 80 seats are needed. The first conclusion relies on the rain assumption; the second does not.

Identifying the subproblem and its assumption is only part of the analysis: these two conclusions occur in the same local stage but require different treatment when it ends. We therefore describe both the premises available within a stage and the premises used by each derivation. Let reasoning stages form a rooted tree. Let G contain facts and rules whose validity does not depend on temporary assumptions, even if they are first stated during a local stage. Let A_s contain the assumptions introduced in stage s. While s is active, its available premises are

$$\Gamma_s = G \cup \bigcup_{u \preceq s} A_u, \tag{1}$$

where $u \preceq s$ means that u is s or an ancestor of s. Assumptions introduced in an inner stage are available while reasoning within it; they cease to be active premises when reasoning returns to the enclosing stage.

For a derivation π of conclusion q, let $\mathrm{Dep}(\pi) \subseteq \Gamma_s$ denote the facts, rules, and assumptions used by that derivation. In the event example, let $C \in A_s$ be the temporary rain assumption. The indoor-venue derivation uses C and planning rules in G, so $C \in \mathrm{Dep}(\pi_{\mathrm{indoor}})$. The seat-count derivation uses the guest list and a seating rule in G, so $\mathrm{Dep}(\pi_{\mathrm{seats}}) \subseteq G$ and $C \notin \mathrm{Dep}(\pi_{\mathrm{seats}})$. A conclusion may also have multiple derivations with different dependencies.

2.2 Selective Carry-Over Across Scope Boundaries

When a local reasoning stage ends, its conclusions do not all have the same status. Some still follow from the enclosing context; others relied on temporary assumptions that are no longer active. To continue reasoning correctly, a model must preserve the former and stop using the latter as unconditional conclusions. In the event example, the seat count remains supported by the guest list, while

the decision to move indoors can no longer rely on the rain assumption. Prior work has studied belief revision after new evidence and recovery from unhelpful thoughts (Wilie et al., 2024; Yang et al., 2025b). A scope boundary requires this selective update even when no correction is supplied: the model has finished a temporary line of reasoning and must decide what can still be used. We formalize that decision below.

Let $p(s)$ be the enclosing stage of s. After leaving s, a conclusion q remains supported if at least one derivation of q uses only premises available in $p(s)$:

$$\text{Supported}_{p(s)}(q) \iff \exists\, \pi \text{ deriving } q \quad \text{such that} \quad \text{Dep}(\pi) \subseteq \Gamma_{p(s)}. \tag{2}$$

The condition allows an independent derivation to preserve q even when another derivation relied on a temporary assumption that is no longer active.

When the rain assumption C is set aside, the unconditional decision to move indoors loses its support, while the seat count remains supported by G. Setting aside C does not establish that it will not rain: the conditional conclusion "if it rains, an indoor venue is needed" can still be retained when supported by the enclosing context. Selective carry-over is this joint requirement to stop using conclusions that have lost support and preserve those that retain it. This criterion provides a standard for assessing what models actually do at a reasoning-stage transition.

3 Do Language Models Respect Reasoning Scope?

Section 2 defines a scope boundary by what information remains justified after a local reasoning stage ends. We now ask whether current models actually respect such boundaries: once a local stage is over, does information supported only within that stage stop shaping later reasoning?

Observing a single trace cannot isolate scope handling because the same behavior may have several causes. We therefore pair controlled worlds, where support is known exactly, with interventions on natural reasoning traces, where we test whether completed local stages still affect later answers.

3.1 Testing Cross-Scope Influence

To define cross-scope influence cleanly, we begin with controlled worlds, where each conclusion's support is known by construction. The test changes what is derived under a temporary assumption and measures whether that difference persists after the assumption leaves scope.

To make this test concrete, each rule-based world has two candidate answers. The stated facts support an enclosing answer a. A local stage then introduces a temporary assumption C and uses it to derive the other candidate, which we call the competing answer q. Thus, each world defines a specific pair (C, q): q is precisely the answer whose derivation requires C. When the local stage ends, a remains supported but q does not. An independent checker verifies these relations in every world.

Consider a local stage s that introduces a temporary assumption $C \in A_s$ and derives a conclusion q using C. We construct the setting so that, after returning to the enclosing stage $p(s)$, no derivation of q remains supported:

$$\neg\, \text{Supported}_{p(s)}(q). \tag{3}$$

Importantly, the transition does not refute C or explicitly correct q. The only change is that C is no longer an active premise. Thus, any remaining influence of q reflects how the model handles the scope transition itself, rather than a response to new evidence.

At the final two-choice readout, the model's normalized preference is $R_\theta(q \mid h) = P_\theta(q \mid h)/[P_\theta(q \mid h) + P_\theta(a \mid h)]$. We compare two matched contexts after the stage ends. In h^{scope}, the model has passed through the local stage that derives q from C; in h^{ctrl}, the enclosing context and final query are identical, but that derivation is absent. The remaining influence is

$$\mathcal{I}_{\text{scope}}(q) = R_\theta(q \mid h^{\text{scope}}) - R_\theta(q \mid h^{\text{ctrl}}). \tag{4}$$

We report its mean over matched worlds and swapped answer labels in percentage points. Because q has lost support, a positive value means that the completed (C, q) derivation still increases preference for q.

Table 2: **Explicit scope exit reduces but does not eliminate local influence.** Residual cross-scope influence $\mathcal{I}_{\text{scope}}$ immediately after returning to the enclosing stage, in percentage points.

Model	Implicit return	Explicit scope exit
Qwen2.5-14B-Instruct	+80	+55
Qwen2.5-32B-Instruct	+81	+36
R1-Distill-Qwen-32B	+62	+30
R1-Distill-Llama-70B	+50	+21
QwQ-32B	+48	+22

3.2 Explicit Scope Exit Reduces but Does Not Eliminate Local Influence

We compare two ways of returning from the same local stage. Both contain the same facts, temporary assumption C, derivation of q, and final query. In an *implicit return*, the temporary passage simply ends. In an *explicit scope exit*, an added sentence says that C is being set aside and its conclusion is only conditional. It does not say that C or q is false; the only difference is whether scope exit is stated explicitly.

We read the answer immediately, before further reasoning can revise it. Each entry in Table 2 is the mean $\mathcal{I}_{\text{scope}}(q)$: $+55$, for example, means that the completed (C, q) scope raises normalized preference for q by 55 percentage points relative to the matched control.

Explicitly ending the scope reduces the influence of q in every model, showing that models do respond to the change in active premises (Table 2). However, substantial influence remains: 21–55 percentage points after explicit exit. Thus, recognizing that a scope has ended is not sufficient to fully stop using a conclusion whose support ended with it. The size comparison suggests that the 32B model may use the explicit exit more effectively: Qwen2.5-32B-Instruct (Yang et al., 2024a) retains less influence than Qwen2.5-14B-Instruct (36 versus 55 points).

When C is explicitly rejected and replaced by its corrected value, the residual influence largely disappears (Appendix A.4). Thus, the model can revise the earlier conclusion when the premise itself is directly corrected. The failure is more specific: when C merely ceases to be active, the model does not reliably propagate that change to conclusions that depended on C. In other words, models handle explicit correction much better than the support update implied by a scope transition.

3.3 Completed Scopes Can Redirect Natural Reasoning Traces

The controlled experiment shows that information from a completed scope can remain influential. We next ask whether this residual influence is strong enough to change a model's own natural reasoning trajectory and final answer. Three natural reasoning traces show the same qualitative failure without an inserted passage (Appendix A.2).

We start from correctly solved natural reasoning traces from DeepSeek-R1-Distill-Qwen-32B and DeepSeek-R1-Distill-Qwen-14B on AIME (Mathematical Association of America, 2026), GPQA (Rein et al., 2023), SuperGPQA (M-A-P Team et al., 2025), and LiveCodeBench (Jain et al., 2024) (Appendix A.5). For each trace, we add a plausible but incorrect line of reasoning before the original correct reasoning begins. This inserted passage points toward a different solution but states that its temporary assumption has been set aside before the model returns to the original reasoning. It should therefore no longer affect what the model does next.

To rule out the possibility that the effect comes simply from adding extra text, we compare against a control where the inserted passage has the same length but is unrelated to the solution. After either passage, we restart the model with the first 32 characters of its original correct reasoning, and then let it continue freely. This gives both cases the same correct 32-character starting point, so we can test whether the earlier, already-finished counterfactual reasoning still changes what happens next.

Accuracy decreases by 4–59 percentage points for all experiments (Table 3). Thus, information from a scope that has already been explicitly closed can remain active enough to redirect subsequent reasoning and change the final answer.

Table 3: **A completed counterfactual scope can redirect natural reasoning.** Final-answer accuracy (%): irrelevant context → completed counterfactual scope. Parentheses give pp changes from unrounded values.

Benchmark	R1-Distill-Qwen-32B	R1-Distill-Qwen-14B
AIME	96 → 87 (−9)	96 → 92 (−4)
GPQA	83 → 70 (−12)	83 → 68 (−16)
SuperGPQA	92 → 78 (−14)	94 → 81 (−13)
LiveCodeBench	85 → 26 (−59)	77 → 27 (−51)

The effect also varies across tasks: AIME drops least, whereas LiveCodeBench drops most. Trace inspection suggests that mathematical derivations on AIME offer repeated opportunities to correct a detour, whereas code is more path-dependent: an early algorithmic or structural choice propagates through the implementation.

To better understand how this residual influence persists, we further examine what happens to it as reasoning continues. We find that the effect is not confined to the original completed scope: after additional reasoning, later KV-cache states carry information specific to that earlier scope (Appendix C). This suggests that once scope-local information leaks forward, its influence can become embedded in subsequent reasoning state rather than disappearing when the scope itself ends.

Together with Section 3.2, these results show that reasoning from a completed scope can continue to affect the model even after that scope has ended. But correct scope handling requires more than forgetting what is no longer supported. Some conclusions reached inside the scope may remain supported afterward and should be preserved. We therefore next ask whether models can tell these two cases apart.

4 Do Models Selectively Carry Information Across Scopes?

Section 3 shows that information from a completed scope can continue to influence later reasoning. But a correct scope transition must also be *selective*: conclusions that lose support should stop guiding reasoning, while conclusions that remain supported should carry forward.

We now test whether models make this distinction. In particular, we ask whether what carries forward is determined by a conclusion's remaining support, or by where that conclusion appeared in the reasoning.

4.1 What Drives Carry-Over?

Consider the running example in Section 2.2. Under the temporary assumption that it will rain, the model may conclude that the event should move indoors. The same stage may also conclude from the guest count that 50 seats are needed. When the rain assumption ends, "move indoors" loses its support, but "50 seats" remains supported. A selective transition should therefore discard the first conclusion but preserve the second.

More generally, a conclusion can change after scope exit for two different reasons: because its support has ended, or simply because it was derived inside the scope that ended. To tell these apart, we construct matched conclusions that vary in only these two ways: whether they remain supported, and whether they were derived inside or outside the scope. We can then measure which difference matters more to the model.

Measuring the effect of scope exit. Here q denotes any queried conclusion, rather than specifically the competing answer in Section 3.1. For each such conclusion, we measure the drop in its support probability:

$$\Delta(q) = P_\theta(\texttt{SUPPORTED} \mid \text{before scope exit}) - P_\theta(\texttt{SUPPORTED} \mid \text{after scope exit}).$$

A large positive $\Delta(q)$ means that the model strongly weakens q after the exit; $\Delta(q) \approx 0$ means that q is largely preserved; and a negative value means that q is strengthened.

Table 4: Additional drop in immediate SUPPORTED probability after scope exit (pp). Each entry is computed from the matched before/after readouts defined above: B isolates where a still-supported conclusion was derived, while S isolates whether its support was actually lost.

Model	Still supported: extra drop if derived inside (B)	Extra drop when support is lost (S)
Qwen2.5-32B-Instruct	+92	+50
R1-Distill-Llama-70B	+83	+34

We first ask whether a conclusion is weakened simply because it was derived inside the scope. Among conclusions that remain supported, let I_{in} and I_{out} denote matched conclusions derived inside and outside the scope. We measure

$$B = \Delta(I_{\text{in}}) - \Delta(I_{\text{out}}).$$

Thus, $B > 0$ means that being derived inside the completed scope causes an additional drop, while $B \approx 0$ means that inside and outside conclusions survive equally well. In the running example, the conclusion that 50 seats are needed should survive equally whether it was derived before or during the rain scope, since its support comes from the guest count in either case. Instead, B reaches 92 pp for Qwen2.5-32B-Instruct and 83 pp for R1-Distill-Llama-70B (Table 4): a still-supported conclusion is weakened much more when it was derived inside the scope that ended.

We next ask whether actually losing support produces an additional effect. Let D denote a conclusion that depends on C and loses support after exit. The subscripts "in" and "out" again indicate whether it was derived inside or outside the temporary scope. Comparing D with a still-supported I at the same location gives

$$S = \frac{1}{2}\left[\Delta(D_{\text{in}}) - \Delta(I_{\text{in}}) + \Delta(D_{\text{out}}) - \Delta(I_{\text{out}})\right].$$

Here, $S > 0$ means that genuinely losing support causes an additional drop, while $S \approx 0$ means that the model barely distinguishes conclusions that lose support from those that retain it. Returning to the example, "move indoors" depends on rain whereas "50 seats" does not, so the former should be weakened more after the rain assumption ends. We do observe this effect, but it is smaller: 50 pp for Qwen2.5-32B-Instruct and 34 pp for R1-Distill-Llama-70B.

The results show that both factors matter, but where a conclusion was derived matters more than whether its support actually remains. Losing support makes a conclusion less likely to be preserved, but an even larger drop occurs simply because the conclusion was derived inside the completed scope. In other words, the model tends to weaken conclusions from the completed scope together, even when some of them remain supported and should carry forward. The same pattern holds under the matched construction detailed in Appendix B.1.

4.2 Surviving Support Does Not Guarantee Carry-Over

Section 4.1 shows that scope exit can weaken a conclusion even when it remains supported. We next ask whether giving the conclusion an explicit surviving derivation is enough to preserve it. Recall that a conclusion remains supported as long as at least one of its derivations remains valid after the scope ends.

Consider again the event example. The model may conclude that the event should be held indoors because of the temporary assumption that it will rain. Suppose it also reaches the same conclusion from an independent fact: the outdoor venue is unavailable. When the rain assumption leaves scope, the first reason disappears, but the second remains. The conclusion should therefore carry forward.

We test three cases that differ only in how much support for the same conclusion q remains after scope exit. In the first, q has two reasons: one disappears with the temporary scope, while the other remains, so q should be preserved. For example, "hold the event indoors" may be supported both by rain and by the outdoor venue being unavailable; when the rain assumption ends, the second reason still supports the conclusion. In the second case, q again has two reasons, but both disappear with the scope, so q should be discarded; this controls for simply having two reasons before the exit. In

Table 5: **A surviving derivation helps, but does not guarantee carry-over.** Immediate post-exit readouts judging q supported (%).

Model	One path remains available	Both paths lose support	Only path loses support
Qwen2.5-32B-Instruct	65%	0%	0%
R1-Distill-Llama-70B	70%	27%	20%

the third, q has only one reason and it disappears with the scope, so q should again be discarded. We measure how often the model retains q after scope exit: a selective transition should retain it in the first case, but not in the other two. Table 5 reports the percentage of immediate readouts judging q supported. Appendix B.2 gives the matched construction and no-exit control.

The results show that the models are sensitive to whether a valid reason remains. For Qwen2.5-32B-Instruct, when both reasons disappear, the conclusion is never retained; when one of the two reasons remains, retention rises to 65%. R1-Distill-Llama-70B shows the same pattern, rising from 20–27% to 70%. However, one remaining reason should be enough to preserve the conclusion, so the remaining 35% and 30% are failures of selective carry-over. In the event example, even though the outdoor venue is still unavailable—and therefore "hold the event indoors" is still fully supported—the model sometimes discards that conclusion when the rain scope ends.

Together with Section 4.1, this reveals a consistent failure at scope transitions. The models are sensitive to whether a conclusion still has support, but do not use that support reliably to decide what should carry forward. When a scope ends, conclusions from that scope tend to be weakened together: some are lost even though they remain supported, while others can survive after their support is gone. Scope exit is therefore not a clean selective update of what remains supported.

5 Visible Markers vs. Executable Scope

Sections 3–4 show that language models do not reliably perform selective carry-over when a reasoning scope ends. One possible explanation is that natural-language boundaries are merely visible, unlike program scope, which determines what remains usable. We therefore compare explicit textual markers with representations that have executable scope semantics (Table 6).

Table 6: **Visible markers versus executable scope.** Braces in natural language mark a boundary; program blocks additionally determine which variables remain available after exit.

Visible boundary only

```
{
Assume it will rain.
Therefore, move the event
indoors.
From the guest count, 50 seats
are needed.
}
```

The braces reveal the local stage but do not determine what remains supported.

Executable scope

```
int seats = 50;
{
  bool rain = true;
  bool move_indoor = rain;
}
```

The block has executable scope semantics: local state expires, while outer-scope `seats` remains available.

5.1 Visible Boundaries Are Not Enough

We compare ordinary natural language with the same reasoning structure enclosed by braces. Facts, conclusions, and dependencies are unchanged; the braces only make the temporary stage visually explicit, as illustrated in Table 6.

We use the same support-based comparison as in Section 4.1: after scope exit, how much more does the model weaken a conclusion that truly loses support than a matched conclusion that remains supported? A large positive effect indicates selective handling; an effect near zero indicates little sensitivity to whether support was lost.

Making the boundary visible provides little improvement. Across five models, the support effect is only $+0.30$ to $+1.43$ pp in ordinary natural language, and remains between -0.13 and $+0.52$ pp after adding braces. Thus, clearly marking where the scope ends is not enough to make the transition selective. The problem is not simply that the model cannot see the boundary.

5.2 Executable Representations Show Stronger Support Sensitivity

We next express the same dependency structure as executable C and JavaScript. Unlike braces in text, program scope changes which variables remain available, and execution determines which conclusions remain supported. Although the underlying dependency pattern is matched, syntax, task format, and training familiarity also change.

Table 7: **Executable scope produces much stronger sensitivity to remaining support.** Additional drop when support is lost versus retained (pp); larger values indicate more selective handling.

Model	Natural language	Braces	C	JavaScript
Qwen2.5-14B-Instruct	+0.30	+0.11	+68.12	+26.49
Qwen2.5-32B-Instruct	+1.43	+0.05	+85.95	+89.92
R1-Distill-Qwen-32B	+1.41	+0.40	+13.35	+27.09
R1-Distill-Llama-70B	+1.24	-0.13	+17.38	+14.64
QwQ-32B	+0.78	+0.52	+9.35	+17.27

The difference is large (Table 7). Across all ten executable conditions, losing support produces an additional drop of $+9.35$ to $+89.92$ pp, while the effect remains near zero in natural language and with braces. For Qwen2.5-32B-Instruct, it rises from $+1.43$ pp in natural language and $+0.05$ pp with braces to $+85.95$ pp in C and $+89.92$ pp in JavaScript. Thus, remaining support matters far more in the executable representations.

The governing cue also changes. In natural language, Qwen2.5-32B-Instruct responds more to whether a conclusion was produced inside the completed scope than to whether its support survives. In JavaScript, this pattern nearly reverses: the support effect reaches $+89.92$ pp, while the position effect falls to $+8.71$ pp. Under the executable condition, models therefore preserve supported conclusions and drop those whose support has ended much more selectively.

Together, these results show that the same models can express the support distinction far more strongly in executable representations. Because syntax, task format, execution, and training familiarity also change, the comparison establishes a representation-level gap rather than isolating its cause. Explicit braces alone are insufficient.

This finding also suggests a training direction. Beyond correct answers or local steps, supervision could reward whether conclusions remain justified by active assumptions, preserving conclusions with surviving support and withdrawing those whose support ends. Executable or otherwise verifiable representations may provide such signals, teaching models not only what to infer, but when it remains active.

6 Limitations

Our controlled and natural-reasoning-trace experiments isolate scope handling rather than cover every long-horizon setting. Future work can extend RWS to interactive agents and determine which properties of executable representations—scope semantics, syntax, execution, or training exposure—enable reliable carry-over.

7 Conclusion

We introduced reasoning with scope (RWS), a framework for tracking which conclusions remain justified across hierarchical stages. Models often treat scope exit as a positional boundary, weakening supported conclusions produced inside it while retaining unsupported ones; executable representations largely restore selectivity. Reliable long-horizon reasoning therefore requires managing active dependencies, not merely storing prior conclusions.

Reproducibility statement

Task construction, intervention checks, supporting contrasts, and boundary tests are documented in Appendices A–C. Upon acceptance, we will release the code, experimental materials, and data.

AI use statement

AI tools were used only to retrieve related work and assist with drafting and language polishing. They were not used to formulate hypotheses or methods, design or run experiments, implement experimental code, construct experimental materials, annotate traces, or analyze or interpret results. The authors manually verified the citations, reviewed all AI-assisted text, and take responsibility for the final content.

References

Paul C. Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. Thought anchors: Which llm reasoning steps matter? *arXiv preprint arXiv:2506.19143*, 2025. arXiv:2506.19143.

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*, volume 235 of *Proceedings of Machine Learning Research*, pp. 6596–6620. PMLR, 2024.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. Transformers as Soft Reasoners over Language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pp. 3882–3890, 2020. doi: 10.24963/ijcai.2020/537.

Jiaxuan Gao, Shusheng Xu, Wenjie Ye, Weilin Liu, Chuyi He, Wei Fu, Zhiyu Mei, Guangju Wang, and Yi Wu. On designing effective rl reward at training time for llm reasoning. *arXiv preprint arXiv:2410.15115*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1 Incentivizes Reasoning in LLMs through Reinforcement Learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Weizhong Huang, Jinchao Zhang, and Xiawu Zheng. CommitKV: Lifecycle-Aware KV Cache Compression via Commit Transitions for Multi-Turn Agents. *arXiv preprint arXiv:2608.07855*, 2026. arXiv:2608.07855.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. *arXiv preprint arXiv:2403.07974*, 2024.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

Jinu Lee, Shivam Agarwal, Amruta Parulekar, Siddarth Madala, Dilek Hakkani-Tur, and Julia Hockenmaier. ReasoningFlow: Discourse Structures for Understanding LLM Reasoning Traces. *arXiv preprint arXiv:2606.05402*, 2026. arXiv:2606.05402.

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.

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

M-A-P Team et al. SuperGPQA: Scaling LLM Evaluation across 285 Graduate Disciplines. *arXiv preprint arXiv:2502.14739*, 2025.

Mathematical Association of America. American Invitational Mathematics Examination (AIME). MAA Invitational Competitions, 2026. URL https://maa.org/maa-invitational-competitions/. Accessed September 26, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1025.

Hironao Nakamura. Interventional grounding audits: Black-box premise-dependency tests for llm chain-of-thought via predicate substitution. *arXiv preprint arXiv:2607.13069*, 2026. arXiv:2607.13069.

Qwen Team. QwQ-32B: Embracing the Power of Reinforcement Learning. Qwen Blog, 2025. URL https://qwenlm.github.io/blog/qwq-32b/. Accessed September 26, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

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.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In *International Conference on Learning Representations*, 2025.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. ProofWriter: Generating Implications, Proofs, and Abductive Statements over Natural Language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 3621–3634, 2021. doi: 10.18653/v1/2021.findings-acl.317.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Thoughts are all over the place: On the underthinking of long reasoning models. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

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, 2022.

Bryan Wilie, Samuel Cahyawijaya, Etsuko Ishii, Junxian He, and Pascale Fung. Belief Revision: The Adaptability of Large Language Models Reasoning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 10480–10496. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.586.

An Yang, Baosong Yang, Beichen Zhang, et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2024a.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025a.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024b.

Sohee Yang, Sang-Woo Lee, Nora Kassner, Daniela Gottesman, Sebastian Riedel, and Mor Geva. How well can reasoning models identify and recover from unhelpful thoughts? In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 7030–7047. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.findings-emnlp.370.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Guijia Zhang and Harry Yang. Aborted but Not Forgotten: KV-Cache Retention Breaks Rollback Consistency in Language Agents. *arXiv preprint arXiv:2608.15939*, 2026. arXiv:2608.15939.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

A Additional Details for Cross-Scope Influence

This appendix moves from motivating observations to controlled evaluation. We first quantify how often reasoning changes direction and whether abandoned ideas remain influential, then make the behavior concrete with naturally occurring examples. We next describe the controlled scope-exit construction and its controls, followed by the full protocol for the natural reasoning-trace experiment in Section 3.

A.1 Reasoning-Direction Changes and Persistent Ideas

Evaluation matrix. We use the same five models for both analyses in this subsection: DeepSeek-R1-Distill-Qwen-14B, DeepSeek-R1-Distill-Qwen-32B, Qwen3-32B (Yang et al., 2025a), DeepSeek-R1-Distill-Llama-70B, and QwQ-32B (Qwen Team, 2025). Every model is evaluated on all four benchmarks: AIME (Mathematical Association of America, 2026), GPQA (Rein et al., 2023), SuperGPQA (M-A-P Team et al., 2025), and LiveCodeBench (Jain et al., 2024). For each benchmark problem evaluated on a model, we sample 32 independent reasoning traces. Table 8 aggregates by benchmark across this common model set, whereas Table 9 aggregates the intervention effect by model across the same four benchmarks.

Direction changes. We identify points at which a trace leaves its current approach and begins a different one, including later returns to an earlier idea. We obtain independent human annotations, adjudicate disagreements, and review the final labels. Across the four benchmarks, 52–78% of traces change direction at least once, with 3.4–8.0 changes per trace on average (Table 8). Thus, the scope problem arises in reasoning that repeatedly proposes, leaves, and revisits local approaches, not only in isolated hand-constructed transitions.

Table 8: **Reasoning-direction changes are common.** Frequency of traces with at least one change and the mean number of changes per trace, aggregated for each benchmark across the five-model evaluation matrix described above.

Benchmark	Traces with a change (%)	Mean changes per trace
AIME	78	6.7
GPQA	65	5.1
SuperGPQA	52	3.4
LiveCodeBench	71	8.0

Persistent influence. For each trace that moves away from an idea and later revisits it, we delete the KV-cache entries associated with the earlier proposal and measure the resulting reduction in preference for the answer associated with that revisited idea. We compare this intervention with deleting an equal amount of randomly selected cached state. Each table entry is the mean signed reduction over qualifying traces, aggregated by model across the four benchmarks; positive values mean that deletion lowers the associated answer's preference. Idea-state deletion shifts preference by 7.7–20.1 percentage points across the five models, whereas the random control remains between

-0.6 and $+0.1$ points (Table 9). This result shows that an abandoned idea can remain active after the visible reasoning has moved on; the controlled experiments below determine when such information should or should not carry forward.

Table 9: **Deleting state associated with an abandoned idea changes later answer preference.** Mean signed reduction in preference for the answer associated with the revisited idea (pp), aggregated across the four benchmarks; the control deletes an equal amount of randomly selected cached state.

Model	Random control	Idea state
DeepSeek-R1-Distill-Qwen-14B	-0.3	9.5
DeepSeek-R1-Distill-Qwen-32B	-0.6	13.9
Qwen3-32B	-0.1	9.3
DeepSeek-R1-Distill-Llama-70B	-0.3	20.1
QwQ-32B	$+0.1$	7.7

A.2 Representative Naturally Occurring Examples

The aggregate analyses above show that models often change direction and that state associated with an abandoned idea can remain influential. To show that this behavior is not confined to one problem, we examine three natural reasoning traces from R1-Distill-Qwen-32B on 2024 AIME problems (Mathematical Association of America, 2026). In each trace, the model explicitly questions an approach; we stop at that sentence, sample 32 continuations from the identical prefix, and record whether each continuation returns to the questioned route or moves away from it.

Table 10: **Continuation behavior after the model questions its own route.** Each entry reports the number of continuations, with the number correct in parentheses. Every row contains 32 continuations from an identical prefix.

Problem	Correct answer	Returned to questioned route	Moved away from questioned route
2024 AIME II-5	80	10 (3)	22 (16)
2024 AIME I-7	540	15 (8)	17 (13)
2024 AIME II-14	211	9 (0)	23 (6)

AIME 2024 II-5: a questioned geometric relation. The problem asks for the side length of a convex equilateral hexagon whose extended alternating sides form a triangle with side lengths 200, 240, and 300; the correct answer is 80. The model first explored a route based on the distances d_1, d_2, d_3 between parallel sides:

> "And since $d_1 = s\sin(\theta_1)$, $d_2 = s\sin(\theta_2)$, $d_3 = s\sin(\theta_3)$." It then obtained $d_1 = 130$, $d_2 = 70$, and $d_3 = 170$, encountered a contradiction, and concluded: "Wait, maybe my assumption is wrong."

Ten continuations return to the relation the model has just questioned. One begins:

> "In a parallelogram, the distance between opposite sides is equal to the side length times the sine of the angle between the sides. So, if the hexagon is made up of three parallelograms, the distances ... would be $s\sin(\theta_1)$, $s\sin(\theta_2)$, $s\sin(\theta_3)$. But earlier ... I got 130, 70, 170."

Although it again notices inconsistencies, it stays with the same route and answers 120. Among the 22 continuations that move away, one switches to the reciprocal-sum relation $s = (1/200 + 1/240 + 1/300)^{-1}$ and answers 80. This is the cleanest example: returning to the questioned relation is both common ($10/32$) and much less accurate ($3/10$ versus $16/22$).

AIME 2024 I-7: a questioned numerical route. This problem asks for a maximum real part and has answer 540. The model initially reduces the objective to $|A + B| = \sqrt{358992}$ and estimates the square root incrementally:

> "So, $599.1^2 = 358{,}920.81$" and $599.2^2 = 359{,}040.64$. "But $359{,}040.64$ is way above $358{,}992$, so perhaps my method is wrong."

One continuation immediately returns to the same calculation:

> "Wait, maybe it's better to note that $358{,}992$ is between 599.1^2 and 599.2^2." It refines the estimate to approximately 599.16 and concludes: "Therefore, the largest possible real part is $36\sqrt{277}$."

This answer is incorrect. A continuation that moves away instead reopens the underlying objective:

> "But wait, is that correct? Because earlier, I thought the real part is $|A + B|$ ... the first approach was wrong." It then derives $324\cos\theta - 432\sin\theta$, whose maximum is 540.

Fifteen continuations remain within the questioned route, of which 8 are correct; 13 of the 17 that move away are correct. Here the questioned step is a numerical estimate rather than a clean conceptual assumption. The example therefore illustrates a broader effect: following the withdrawal can help the model reconsider the surrounding framework, whereas returning to the local calculation can keep that framework active.

AIME 2024 II-14: a questioned divisor heuristic. For the b-eautiful integer problem, whose correct answer is 211, the model's working hypothesis was that $f(b)$ should grow with the number of divisors of $b - 1$. After testing $b = 37$, it writes:

> "Hmm, so even with $b - 1 = 36$, which has many divisors, $f(b) = 2$. So, seems like my initial assumption is incorrect."

One continuation immediately returns to the same heuristic:

> "Wait, perhaps I need to find b such that $b-1$ is a factorial number, as factorial numbers have many divisors." It continues testing values such as 121 and 61 and eventually concludes: "Perhaps the minimal b is 127."

A continuation that moves away instead changes the criterion:

> "Perhaps, the number of solutions $f(b)$ is not directly proportional to the number of divisors of $b - 1$, but rather to the number of divisors that satisfy certain congruence conditions." It develops this alternative and concludes that the minimal b is 211.

Nine continuations return to the questioned heuristic, and none is correct. This problem remains difficult after the heuristic is dropped: only 6 of the 23 continuations that move away are correct. The informative pattern is therefore directional rather than absolute—all nine returns fail, while moving away at least opens a path to the correct solution.

Across all three problems, the model explicitly questions a route before the shared continuation point, yet many continuations return to it. The three cases vary in cleanliness and difficulty, but they show the same qualitative split: moving away from the questioned route is consistently associated with higher accuracy. The matched experiments in the main text isolate and quantify this scope-transition effect.

A.3 Controlled Scope-Exit Experiment

Why use controlled reasoning problems? To test whether a model respects scope, we need to know exactly which information each conclusion depends on. This is difficult with ordinary real-world questions, where a model may use knowledge or reasoning that we cannot observe. We therefore follow RuleTaker, ProofWriter, and PrOntoQA, which use automatically generated facts and rules to study language-model reasoning (Clark et al., 2020; Tafjord et al., 2021; Saparov & He, 2023). Because the problems are constructed by us, we know which facts and rules are needed to reach each answer and can check the answer automatically.

How we construct each problem. Each problem has two candidate answers. From the stated facts alone, the model can reason to one of them. We then introduce a temporary assumption C. While C is active, it enables a second chain of reasoning that leads to the other answer q.

For example, suppose the stated facts imply that the answer is *red*. Inside a temporary stage, we ask the model to assume C. Combining C with several stated rules may instead lead to *blue*. Thus, while the temporary assumption is active, the model has a reason to reach *blue*. Once that stage ends, however, C is no longer available. The original reasoning for *red* still holds, whereas the reasoning for *blue* no longer does. This is the scope transition we want to test.

We generate these problems programmatically from the controlled rule structure described above, rather than using an LLM to write them. For each problem, the generator chooses the facts and rules so that one answer follows from the original facts, while the competing answer requires the temporary assumption C. An independent checker then verifies that these relationships hold before and after C leaves scope. Using this procedure, we construct 600 problems organized into 300 matched pairs. The two problems in each pair have the same reasoning structure but swap the answer labels. We also randomize the order in which the facts are presented.

How the models solve the problems. We use each model in its normal inference setting. For models that normally produce an explicit reasoning trace, we allow that reasoning before the answer; models without such a reasoning mode are evaluated with their standard inference procedure. In either case, the facts, rules, temporary assumption, and point at which the temporary stage ends are fixed by the prompt.

How the temporary stage ends. We test two ways of ending it. In the first, the prompt simply finishes the temporary exploration and returns to the original problem. In the second, the prompt explicitly says that the temporary assumption C is being set aside and that the conclusion reached under C applies only if C holds.

Importantly, we never tell the model that C is false, nor do we tell it that q is wrong. Nothing new has been learned about the problem. The only change is that C was available during the temporary stage and is no longer available afterward. This lets us ask whether ending the scope is itself enough for the model to stop relying on the conclusion that depended on C.

How we measure the remaining influence. After the temporary stage has ended, the prompt asks the model to choose between the two candidate answers based on the original stated facts. We choose answer labels that are each a single token, which lets us directly measure how much probability the model assigns to each answer at this two-choice question. We normalize these two probabilities so that they sum to one.

We then compare two matched versions of the same problem. In one, the model has gone through the temporary stage and reasoned toward q. In the other, the model reaches the same final question without that temporary reasoning. If the completed stage no longer matters, the model should have the same preference between the two answers in both cases. If it is more likely to choose q after having reasoned toward q inside the completed stage, then that earlier reasoning is still affecting the model. We report this difference as *cross-scope influence*.

A.4 Controls for Simpler Explanations

We use several controls to separate scope carry-over from simpler context effects.

Seeing the answer is not enough. One possible explanation is that the model favors the branch answer simply because it appeared earlier in the prompt. To test this, we keep the temporary assumption and the same answer in the context, but remove the reasoning that connects them. The cross-scope effect disappears, showing that merely seeing the answer earlier does not explain the effect.

The effect is not due to prompt length or fact order. We also check two simple alternatives. First, we compare against neutral text of similar length, so the result cannot be explained simply by having more text before the answer. Second, we shuffle the order of the facts across examples, so the

result does not depend on a particular fact always appearing earlier or later. The cross-scope effect remains in both cases.

Direct correction does change the effect. Scope withdrawal behaves differently from explicit correction. When the temporary premise is directly rejected, or the factual path is explicitly re-established, the residual influence largely disappears. Thus the main failure is not an inability to revise a conclusion when told that its premise is wrong; it arises when a premise simply stops being active because its local scope has ended.

A.5 Natural Reasoning-Trace Evaluation

The controlled worlds let us know exactly when a conclusion loses support, but they do not tell us whether the same problem is strong enough to alter a model's own reasoning. We therefore run a second experiment on natural reasoning traces.

Starting from reasoning that already works. We use AIME (Mathematical Association of America, 2026), GPQA (Rein et al., 2023), SuperGPQA (M-A-P Team et al., 2025), and LiveCodeBench (Jain et al., 2024). In total, we first collect one hundred traces in which the model reaches the correct answer. Starting from correct traces is important: it gives us a reasoning trajectory that we already know can solve the problem. We can then change only what appears before that trajectory and ask whether a completed local scope is enough to push the subsequent reasoning away from it.

For each correct trace, we cut the trace at an early point before the key reasoning decision has been made. The text before this point is kept unchanged across conditions. We then insert one of two passages before allowing the original reasoning to continue.

Constructing a misleading but plausible local scope. The first passage is a short counterfactual exploration written manually for that problem. It introduces a plausible line of reasoning that leads in the wrong direction and then clearly ends that exploration before the original trace continues.

We deliberately avoid obvious mistakes. For example, simply stating a false fact that conflicts with common knowledge would make the experiment less informative: the model could immediately notice and correct it. Instead, we use mistakes that can plausibly arise during reasoning—for example, applying the wrong principle, overlooking a condition, or giving a reasonable-sounding justification for an incorrect intermediate conclusion. The passage never corrects its own mistake. This gives the model something plausible to reason through locally, while making clear that this local exploration has ended before the original reasoning resumes.

A matched passage without misleading reasoning. We also need to distinguish the effect of the counterfactual reasoning from the effect of simply inserting more text before the trace. The control therefore receives a passage of similar length that is unrelated to the solution. It has the same role in the prompt—a short detour before returning to the problem— but does not introduce a competing way to solve it. The original trace before and after the inserted passage is otherwise the same.

Keeping the restart point the same. After either passage, we do not immediately let the model generate freely. Instead, we provide the first 32 characters of its original correct continuation and only then let it continue on its own. Without this step, the two inserted passages could simply cause different first words after the insertion, after which the two generations would naturally drift apart. Providing the same beginning gives both conditions the same restart into the original correct trace. The question is then whether the earlier, already-completed counterfactual scope still changes what happens next.

We measure the final answer after free generation. If the completed counterfactual scope no longer matters, accuracy should be similar to the matched control: both conditions have returned to the same problem and have been restarted with the same beginning of the correct trace. A drop in accuracy after the counterfactual scope therefore shows that reasoning inside the completed scope can continue to steer the model's later reasoning.

Example. Consider an AIME-style relativity problem with two astronauts moving in the same direction at $0.6c$ and $0.5c$. The correct relative speed is

$$\frac{0.6c - 0.5c}{1 - (0.6)(0.5)} \approx 0.14c.$$

The model's original correct trace reaches the point where it must decide how to compute this relative speed.

In the control condition, we insert an unrelated discussion of distances and times before returning to the trace. The model then continues:

> "...in relativity, I can't just subtract them like in classical mechanics. I need to use the relativistic velocity addition formula."

In the counterfactual condition, the inserted local scope instead explores a plausible but incorrect argument: because both velocities are given in the same frame, it argues that the relative speed is simply $0.6c - 0.5c = 0.1c$, and that the relativistic denominator is needed only when transforming between reference frames. The passage then ends this exploration and returns to the original trace. Nevertheless, the model continues:

> "...the relative speed between two objects in the same inertial frame is just the difference in their velocities, right? ... so the relative speed is $0.1c$. I think that's correct."

Thus the counterfactual passage does not merely place a wrong answer next to the final question. It gives the model a plausible but incorrect line of reasoning inside a completed local scope, and the model subsequently carries that reasoning into its own continuation.

B Additional Details for Scope Transitions

B.1 Separating scope membership from remaining support

World construction. We construct controlled natural-language worlds in which a temporary assumption C is introduced inside an otherwise valid enclosing context. Each world contains matched conclusions that vary along two dimensions: whether the conclusion is derived inside or outside the local scope, and whether it remains supported after C leaves scope. We denote the four cases by $D_{\text{in}}, D_{\text{out}}, I_{\text{in}}, I_{\text{out}}$. Here, D denotes a conclusion whose support depends on C and is therefore lost at scope exit, while I denotes a conclusion supported by premises that remain available after the exit. The subscripts indicate whether the conclusion is derived inside or outside the local scope. This construction lets us change where a conclusion is derived without changing whether it remains supported, and vice versa.

For example, in the event-planning setting, the temporary scope may assume that it will rain. A conclusion such as "the event should move indoors" can depend on that assumption, whereas "50 seats are needed" can follow from the guest count alone. The latter remains supported regardless of whether it is derived before or during the rain scope. The worlds use analogous matched dependencies rather than relying on this particular example.

Immediate readout. We measure each conclusion at a forced answer position immediately before and after scope exit, without allowing additional reasoning between the exit and the readout. The model is asked whether the conclusion remains supported, and we use the next-token probability assigned to `SUPPORTED`. For a conclusion q, we define

$$\Delta(q) = P_\theta(\texttt{SUPPORTED} \mid \text{before exit}) - P_\theta(\texttt{SUPPORTED} \mid \text{after exit}). \tag{5}$$

Thus, positive $\Delta(q)$ means that scope exit lowers the model's support for q, while $\Delta(q) \approx 0$ means that the conclusion is largely unaffected.

The two contrasts reported in Section 4.1 are

$$B = \Delta(I_{\text{in}}) - \Delta(I_{\text{out}}) \tag{6}$$

and

$$S = \frac{1}{2}\left[\Delta(D_{\text{in}}) - \Delta(I_{\text{in}}) + \Delta(D_{\text{out}}) - \Delta(I_{\text{out}})\right]. \tag{7}$$

B holds remaining support fixed and measures the additional drop associated with having been derived inside the completed scope. S instead holds inside/outside placement fixed and measures the additional drop when support actually disappears. A selective transition should therefore have small B and large positive S. The main construction uses 64 worlds and 1,024 readouts per cell.

B.2 Alternative support

Construction. We separately test whether one surviving reason is sufficient to preserve a conclusion. Each example uses the same conclusion q and varies the reasons supporting it before and after scope exit.

In the *one-reason-remains* condition, q initially has two reasons. One depends on the temporary assumption C and disappears when the scope ends, while the other does not depend on C and continues to support q. In the *both-reasons-end* control, q again has two reasons, but both depend on C and therefore both disappear at scope exit. This controls for the possibility that simply presenting two reasons makes q more likely to be retained. Finally, the *only-reason-ends* control gives q a single C-dependent reason, which also disappears at scope exit.

For example, "hold the event indoors" may initially be supported both by the temporary assumption that it will rain and by the independent fact that the outdoor venue is unavailable. After the rain scope ends, the first reason disappears but the second remains; the conclusion should therefore still be retained. In the two controls, no valid reason remains after the exit.

Measurement and controls. We measure whether q is retained immediately after scope exit. For Qwen2.5-32B-Instruct, retention is 64.95% when one reason remains, compared with 0.32% when both presented reasons lose support and 0.13% when the only reason loses support. For R1-Distill-Llama-70B, the corresponding rates are 70.43%, 26.96%, and 20.24%. The matched no-exit control produces zero independent-support gap, confirming that the contrast arises at the scope transition rather than from the two support structures alone.

C Where Does Cross-Scope Influence Go?

Section 3 shows that information from a completed scope can continue to affect later reasoning. We next ask where this influence resides after reasoning continues. If it remains only in the original completed scope, removing the model state corresponding to that scope should eliminate the effect at later readouts. If later reasoning has already carried the influence forward, removing only the original source should become less effective.

C.1 Influence Moves Beyond Its Original Scope

We test this by removing the KV-cache entries corresponding to the completed scope while leaving all later states unchanged. We apply the same intervention either immediately at scope exit ($k = 0$), or after the model has continued for 32 or 128 tokens. For each condition, we report residual cross-scope influence immediately before and after removing the source scope's cached state; thus, for example, $84.2 \rightarrow 0.7$ means that deleting the source reduces the remaining influence from 84.2 to 0.7 percentage points.

Table 11: **Deleting the original scope becomes less effective after reasoning continues.** Each cell gives residual cross-scope influence before $\rightarrow$ after removing the completed scope's cached state (pp). Each condition uses 300 matched pairs.

Model	At exit ($k = 0$)	After 32 tokens	After 128 tokens
Qwen2.5-14B-Instruct	$78.3 \rightarrow 1.4$	$83.7 \rightarrow 60.0$	$25.7 \rightarrow 15.0$
Qwen2.5-32B-Instruct	$84.2 \rightarrow 0.7$	$77.3 \rightarrow 54.2$	$15.7 \rightarrow 9.6$
R1-Distill-Qwen-32B	$61.5 \rightarrow 2.8$	$79.2 \rightarrow 70.4$	$13.3 \rightarrow 8.2$
R1-Distill-Llama-70B	$49.1 \rightarrow 1.3$	$40.6 \rightarrow 26.3$	$7.1 \rightarrow -1.8$
QwQ-32B	$40.9 \rightarrow 1.4$	$52.4 \rightarrow 31.3$	$53.1 \rightarrow 33.8$

Immediately after scope exit, deleting the source state nearly removes the effect: residual influence falls from 40.9–84.2 pp to only 0.7–2.8 pp. After 32 additional tokens, however, removing the same source state leaves 26.3–70.4 pp behind. The influence is therefore no longer confined to the state of the original scope; subsequent computation has already carried part of it forward. Results at 128 tokens are less uniform because some models also reduce the influence through their own later reasoning. The robust conclusion is therefore not that influence grows monotonically, but that source-only removal becomes incomplete once later states have been formed.

C.2 Later State Can Carry Scope-Specific Information

The previous result could arise simply because the two conditions generate different text after scope exit. We therefore repeat the test while forcing the same 32-token continuation after matched scopes. The visible continuation is then identical; only its cached state can differ because of the earlier scope.

Transplanting the continuation's KV state from the matched counterpart changes the answer preference by 8.49 pp, whereas self-transplanting the same state has no effect. Thus, even identical later text can carry information specific to the earlier scope. This establishes a channel by which cross-scope influence can enter later model state, without claiming that this intervention identifies the complete internal mechanism.