NOT ALL THINKING IS CREATED EQUAL: LATENT REASONING DISCOVERS A RECURRENT SEARCH ALGORITHM FOR DEPTH GENERALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models can perform multi-step reasoning and improve task performance through different forms of intermediate computation, from token-based traces to computation carried out in latent space. However, a question remains open: do these different forms of thinking rely on the same underlying mechanism? To address this, we train and compare five variants of the same GPTNeoX backbone from scratch on an extended multi-hop reasoning task (ProsQA-Ext): a vanilla model, a Chain-of-Thought (CoT) model, a Pause Token model, and two latent-reasoning models that are optimized end-to-end without intermediate reasoning traces. We find that, strong in-distribution (ID) performance does not guarantee depth generalization. Vanilla, CoT, and Pause Token models solve ID problems well, but rely largely on local graph features and generalize poorly to out-of-distribution (OOD) problems with longer hops. In contrast, latent variants generalize better and show internal dynamics consistent with forward reachability propagation on the graph. Causal interventions and circuit analysis localize this computation to a sparse recurrent search circuit in the bottleneck latent model: an attention head retrieves graph relations, an MLP and the residual stream update the reachability state across recurrent steps, while multiple attention heads together then do the candidate matching. Together, these results show that different thinking mechanisms can learn distinct computational solutions, even at similar ID performance. In this setting, latent recurrence supports a reusable forward-search algorithm that generalizes beyond the training depth.

1 INTRODUCTION

Prepending "Let's think step by step" to a prompt can improve pretrained language models' performance on reasoning tasks (Kojima et al., 2022; Wei et al., 2022). Models that fail to answer directly can sometimes solve the same question by first generating intermediate steps. More recently, this approach has been successfully scaled by baking the reasoning traces into the training rather than prompting (Chung et al., 2024; Ho et al., 2023; Magister et al., 2023), which enables smaller models to solve problems where CoT prompting alone is ineffective.

However, whether a model actually follows the reasoning traces it generates remains debated. Part of the reasoning traces can be replaced or removed without hurting the final answer (Lanham et al., 2023; Zhao et al., 2026). Models can also benefit from intermediate steps using meaningless filler or pause tokens (Pfau et al., 2024; Goyal et al., 2024). Together, these findings suggest that useful intermediate computation need not be realized as a verbally meaningful reasoning trace. Coconut (Hao et al., 2025) and related works have demonstrated alternative ways to perform such computation by feeding high-dimensional vectors, instead of tokens, directly into the model (Wei et al., 2025). The mechanisms underlying this latent computation remain poorly understood.

Symbolic reasoning tasks were widely used to probe the circuits and computation inside language models (Wu et al., 2025; Brinkmann et al., 2024). Zhu et al. (2025) showed that latent thoughts theoretically can encode multiple search frontiers in superposition and enable parallel search. However, recent work finds that similar patterns also arise in models without recurrence and do not always causally affect the answer (Aswal et al., 2026; Rizvi-Martel et al., 2026), leaving its causal role

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.

contested. More broadly, it remains unclear how the learned computation differs across thinking interfaces, and what mechanisms support generalization beyond the training distribution.

To investigate this question, we focus on five model variants: a vanilla model, a CoT model, a Pause Token model, and two latent-reasoning models based on Coconut. One full-latent model retains access to all previous tokens, while the other bottleneck-latent model can only rely on the intermediate hidden representations when generating answers. We train these five models on an extended version of the well-established ProsQA task. Notably, the latent variants are trained without intermediate reasoning traces and RL, allowing us to examine whether latent reasoning can discover a generalizable reasoning mechanism without being shown how to solve the task step by step and without slow trial-and-error process.

By testing these models on out-of-distribution (OOD) problems, we find that strong performance within the training range does not guarantee depth generalization, with the latent variants performing best on OOD problems. The strong generalization, together with the lack of shortcut effects in the latent models, indicates that they learn to reason rather than use surface heuristics. Further causal interventions in the bottleneck-latent model show that intermediate states carry intermediate variables during forward search that are reused and transformed across recurrent steps. We localize this computation to a sparse search circuit in which an attention head retrieves graph relations and an MLP, together with the residual stream, update the state for subsequent steps, and finally multiple attention heads read this information for candidate matching. These findings show that different forms of thinking can learn very different computational solutions, even at similar performance. Importantly, latent recurrence supports better discovery of a reusable computation that generalizes beyond the training depth.

2 METHODS

2.1 TASK

A signature of a model that understands rules and can reason with them is that it learns from small scale datasets and generalizes to unseen, more complex problems. In natural language problems, multi-hop symbolic reasoning, such as extended syllogisms, is a good candidate for such datasets: the level of difficulty, i.e., the number of hops, can be controlled, and the symbols used can be permuted without changing the meaning. In this study, adapted from ProsQA by Hao et al. (2025), we construct such a task, ProsQA-Ext. As shown in Fig. 1A, each ProsQA-Ext sample $(x, y) \in \mathcal{D}$ describes a directed acyclic graph (DAG) $G = (V, E)$ and a question about G. The graph description g is a token sequence of premises of the form `A is B.`, each representing a directed edge from A to B. The query q gives a root node $r \in V$ and two candidate nodes $c_0, c_1 \in V$. Together, they form the complete input $x = g \| q$. Exactly one candidate is reachable from r, at a shortest distance of H; the other is either an isolated node or lies on a chain whose root is not r. We denote the reachable candidate by c^* and the reference answer-token sequence by y, which states that r is c^*. At a fixed H, node labels, premise order, and candidate positions are randomly sampled, yielding varied graph structures and inputs x, while the underlying reachability operation f remains unchanged. Unlike ProsQA, we carefully control the generation of G so that no superficial features can be exploited to infer c^* ($\approx 50\%$ accuracy), and we use two separate datasets: a training set with $H \in \{3, \ldots, 6\}$ and a validation set with $H \in \{7, \ldots, 12\}$.

2.2 MODELS

We train five model variants on the same ProsQA-Ext dataset with the same tokenizer to examine how different forms of thinking address the symbolic reasoning problem. All variants use the same GPTNeoX backbone (number of layers=4, hidden size=256, dimensionality of FFN=768). Unlike Hao et al. (2025), all models are trained from random initialization, ensuring their knowledge of the task comes completely through training, rather than possibly inherited from pretraining. The variants differ in their intermediate computation and answer readout (Fig. 1B). For all models, we denote the residual state at token position i after block ℓ by $h_i^{(\ell)}$, and denote $\ell = 0$ as the input to the first block. During prompt encoding, the token embedding E_θ supplies $h_i^{(0)} = E_\theta(x_i)$. After the final block, a final LayerNorm N_θ and an output projection map $h_i^{(L)}$ to next-token logits.

After the input x, the ***Direct*** variant generates the answer directly. The ***Chain-of-Thought*** variant first generates a proof and then the answer. Its training is supervised by both the shortest proofs (a sequence of premises forming the shortest path from r to c^*) and final answers. In the ***Pause-token*** variant, before answer decoding, the model "thinks" by inserting K=6 identical learnable embeddings, $z_t = E_\theta(\texttt{<PAUSE>})$. The two latent variants, ***Full-latent*** and ***Bottleneck-latent***, in their "thinking" process, instead, feed the normalized output of one step directly into the next, instead of decoding it into a token. This process can be described with

$$z_1 = N_\theta\left(h_n^{(L)}\right), z_{t+1} = N_\theta\left(h_{n+t}^{(L)}\right), \qquad 1 \le t < K, \tag{1}$$

in which each vector enters the first block as $h_{n+t}^{(0)} = z_t$, where $n = |x|$.

These two latent variants differ in the readout phase. The ***Full-latent*** model processes z_K with the prompt and recurrent cache retained, keeping both available during answer generation. In this sense, it is similar to the Coconut model (Hao et al. (2025)), but without human guidance involved. However, ***Full-latent*** can still access the g and q during answer decoding, which may allow the model to exploit statistical patterns to solve the problem. To isolate the effect of direct access, we build the ***Bottleneck-latent*** model, which cannot access the recurrent key/value cache and has to re-encode the $z_{1:K}$ as the sole prefix for answer generation. This allows us to examine what is happening inside the latent state trajectories and to decompose the reasoning with memorization.

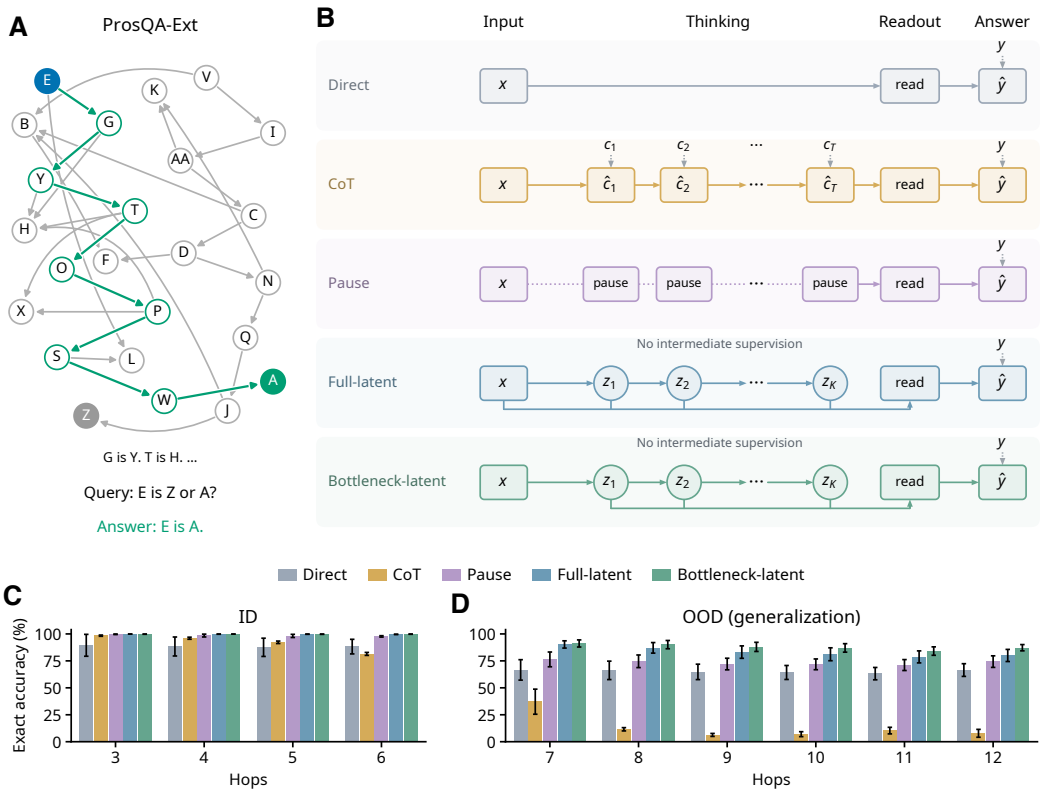

Figure 1: **ProsQA-Ext task and reasoning performance.** **A** shows an example ($H = 8$) with the correct path highlighted in green and distractor edges in gray. **B** shows the five model variants. **C** and **D** show free-generation accuracy on ID and OOD problems, respectively (error bars are SEM).

3 RESULTS

3.1 ID AND OOD DATASET PERFORMANCE

All models are trained on the same set of problems with $H \in \{3, \ldots, 6\}$, using the same budget and optimization settings (see A.2). On the ID validation set, ***Bottleneck-latent*** and ***Full-latent*** achieve nearly perfect accuracy, and ***Pause-token*** is closely behind. ***Chain-of-Thought*** also achieves high accuracy overall, although its performance declines as the H increases. In contrast, without additional computational slots, ***Direct*** is worse than others and is more sensitive to the seeds.

We then test their generalization capability with OOD dataset ($H \in \{7, \ldots, 12\}$). Without further training, the variants start to show divergent behaviors. ***Full-latent*** and ***Bottleneck-latent*** retain the highest accuracy, with ***Bottleneck-latent*** slightly ahead of ***Full-latent***, and both clearly outperforming ***Pause-token*** and ***Direct***. Unexpectedly, ***Chain-of-Thought*** performs worst despite its strong ID performance. Thus, models that perform similarly on the training range can generalize very differently beyond it (Fig. 1C,D).

3.2 HOW DO DIFFERENT MODELS SOLVE THE PROSQA-EXT TASK?

3.2.1 ALIGNMENT WITH FORWARD PROPAGATION

The divergent OOD performance suggests that these variants may reach the same answer through different computations. Two natural heuristic strategies are forward propagation from the query root and backward tracing from the candidate answers. We first test whether their representations track forward propagation. Using representational similarity analysis (RSA), we compare pairwise dissimilarities between model representations and algorithmic states, without assuming a one-to-one correspondence between model steps and algorithmic updates (Appendix A.3).

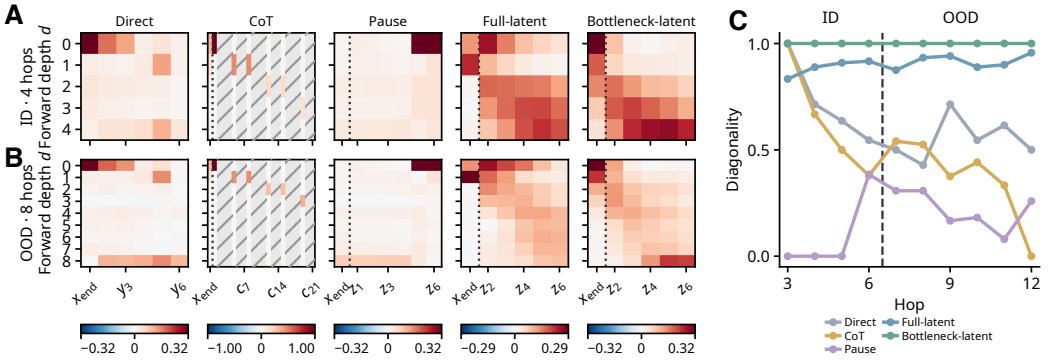

Figure 2: **Representational alignment with forward graph propagation.** **A** and **B** show RSA heatmaps for five variants on ID (4-hop) and OOD (8-hop) problems, respectively, using the same examples across variants. Each entry shows the Spearman correlation between pairwise model-representation dissimilarities and pairwise graph-frontier dissimilarities at depth d. For $G = (V, E)$ with query root r, the propagation frontiers are defined by $F_0 = \{r\}$ and $F_{d+1} = \{v \in V : \exists u \in F_d,\ (u, v) \in E\}$. Gray hatched cells denote undefined correlations. Color scales are shared across rows within each variant. **C** shows diagonality across 3–12-hop problems, with colors indicating variants. The dashed line separates ID (3–6 hops) from OOD (7–12 hops).

Direct and ***Pause-token*** show nearly no sequential alignment with forward propagation (Fig. 2A,B). Nevertheless, ***Chain-of-Thought*** shows some alignment, consistent with its supervision on step-by-step proofs, despite very poor OOD performance. The clearest diagonal-like patterns appear in ***Full-latent*** and ***Bottleneck-latent***, suggesting that they learn a forward-search-like procedure without intermediate supervision. We quantify this progression intuition with *diagonality*, which measures whether alignment shifts toward later model positions as algorithm depth increases (Appendix A.4). A score approaching one indicates a consistent progression, without requiring one model step per graph hop. ***Bottleneck-latent*** has the highest and most consistent diagonality across depths, followed by ***Full-latent***, with the clearest separation from the other variants in OOD problems (Fig. 2C). The

corresponding analysis of parallel backward tracing shows no clear, consistent sequential alignment across task depths (Appendix A.3, Fig. A.2).

3.2.2 Local graph shortcuts drive predictions in non-latent models

Latent models represent search-related intermediate variables, but no comparable alignment is found in ***Direct***, ***Chain-of-Thought***, or ***Pause-token***. However, these three variants remained highly accurate within the training distribution, which raises the question of what supports their answers. One possibility is that they exploit shortcuts from local graph features. Such shortcuts are actually available since the graph-generation procedure inherited from ProsQA induces systematic degree asymmetries: correct candidates tend to have lower in-degree than incorrect ones, and the immediate successors of the query root that lead to the correct candidate tend to have lower in-degree and higher out-degree than the alternatives. These correlations provide local predictive cues, so we test whether model variants rely on them by manipulating local graph features.

We first ask whether the models use the in-degree of candidates when generating answers. To test this, we construct a matched-pair dataset, in which we select one or two edges not on the proof path and redirect their destinations to the correct candidate or to the incorrect candidate (Fig. 3A). This matched pair differs only in the in-degrees of two candidates, keeping all others the same. Note that this matched-pair dataset is constructed to isolate individual graph features and differs in structure from the OOD evaluation set in Section 3.1. Evaluating the five variants on this dataset shows that candidate in-degree strongly affects accuracy in ***Direct*** and ***Pause-token*** across the problems with different reasoning depths, but has much smaller effects on the latent models and minimally affects ***Chain-of-Thought*** (Fig. 3B, C).

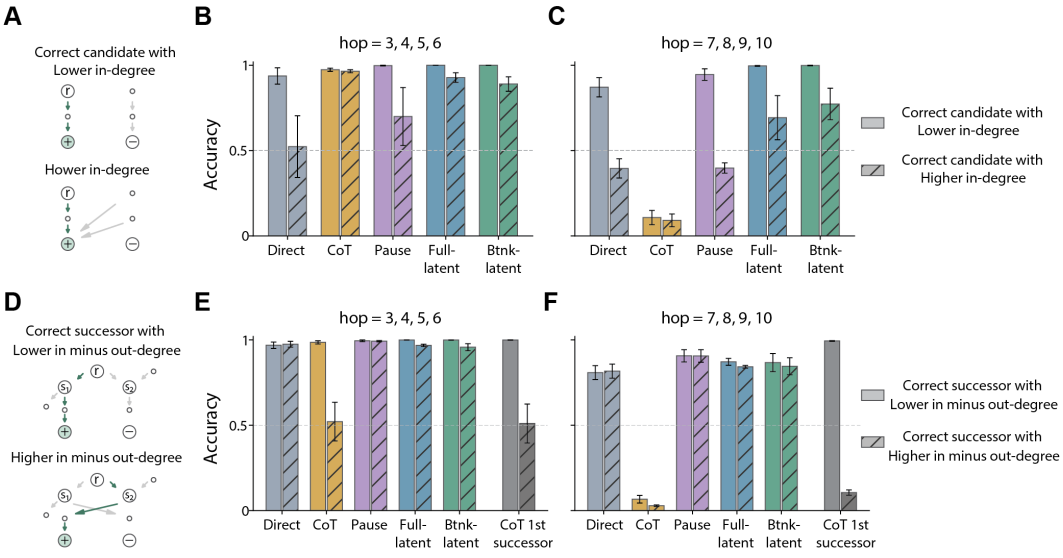

Figure 3: **Effects of candidate and successor degree on model predictions.** **A** shows matched pairs that reverse the candidates' relative in-degree while preserving the proof path and correct answer. **B** and **C** show the final-answer accuracy on problems with short and long reasoning depths. Solid and hatched bars indicate that the correct candidate has lower and higher in-degree, respectively. **D** shows matched pairs that switch which immediate successor of the query root leads to the correct candidate while preserving all node degrees and the correct final answer. **E** and **F** show final-answer accuracy for all five models and first-successor accuracy for CoT on problems with short and long reasoning depths. Solid and hatched bars indicate that the correct successor has a lower or higher in-degree minus out-degree, respectively. In schematics, (r) denotes the query root, (+) and (-) denote the correct and incorrect candidates, and green arrows indicate the proof path.

Next, we ask whether the models use local degree cues at the query root's immediate successors. Unlike other variants, ***Chain-of-Thought*** explicitly identifies a successor in its first generated statement before producing the final answer, making it more vulnerable to the degree at the query root's successors. To test this, we construct a second matched-pair dataset in which the query root has two

immediate successors, with one leading to the correct candidate. Within each pair, we change the in/out degrees of successors by reassigning the source endpoints of non-proof outgoing edges from one successor to the other and redirecting the destination endpoints of non-proof incoming edges between them (Fig. 3D). These manipulations strongly affect both the first-step successor choice and the final answer in ***Chain-of-Thought***, which tends to prefer the successor with lower in-degree minus out-degree, while having little effect on all other variants (Fig. 3E, F).

Together, these results indicate that ***Direct*** and ***Pause-token*** rely substantially on candidate in-degree when predicting the answer, whereas ***Chain-of-Thought*** uses the degree of the query root's successors both when selecting which successor to follow and when predicting the answer. In contrast, the latent models are less affected by either type of local graph structure.

3.3 A RECURRENT SEARCH ALGORITHM IN LATENT REASONING MODELS

3.3.1 SWAPPED CHAINS AS A PROBE OF A "SOFT" FOR-LOOP

Analyses in Fig. 2 suggest that, unlike other variants, latent reasoning models, especially ***Bottleneck-latent***, may implement some recurrent algorithms that support generalization to longer hop problems. To further show if this causally holds, we swap the graph connectivity at each depth and use the change as a probe.

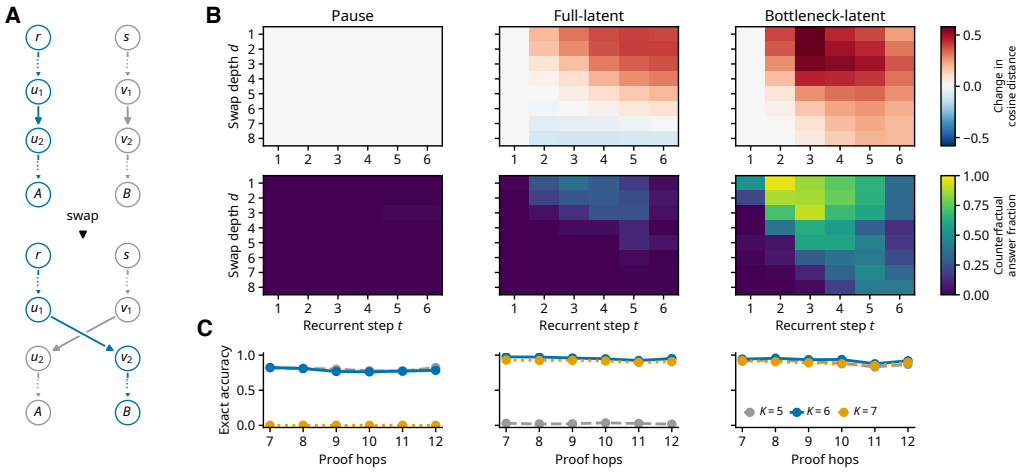

Figure 4: **Controlled interventions probe the recurrent computation in latent reasoning models.** **A** shows how swapping premises shifts the query root r's correct candidate from A to B. In **B**, the top row shows the normalized cosine distance between paired latent states for 8-hop problems with connectivity swaps at depth d, and the bottom row shows the fraction of pairs for which transplanting the latent state at step t redirects the answer to y'. **C** shows OOD accuracy with one fewer ($K = 5$), the trained number ($K = 6$), or one additional ($K = 7$) thinking position. Shaded region indicates 95% CI.

For each sample $x = g\|q$, we build a matched one $x' = g'\|q$ by swapping the destinations of two edges at depth d: one on the solution chain and one on a matched distractor chain. This preserves the query root, node labels, premise order, and node degrees, but switches the reachable candidates, flipping the answer from y to y' (Fig. 4A). Next, we run both inputs to obtain the latent trajectories $z_{1:K}(x)$ and $z_{1:K}(x')$. At step t, we replace $z_t(x)$ with $z_t(x')$, keep the original prompt cache and preceding computation unchanged, and recompute the remaining latent states before generating the answer. We apply this probe to ***Bottleneck-latent*** and ***Full-latent***, with ***Pause-token*** as a control. We measure the state-level effect of the connectivity change as the cosine distance between $z_t(x)$ and $z_t(x')$, normalized by subtracting the distance at $t = 1$ (Fig. 4B top). The causal influence is the fraction of pairs for which transplantation changes the answer from y to y' (Fig. 4B bottom).

If the recurrent computation propagates reachability step by step, shallow perturbations should become effective earlier, and deep ones should influence the output only at later steps. This upper triangular pattern clearly emerges in ***Bottleneck-latent*** (Fig. 4B,C), while it is less clear in ***Full-latent***

and absent in ***Pause-token***. These results suggest that, among the three variants, only ***Bottleneck-latent*** strongly adopts a recurrent algorithm.

Nevertheless, such computation might be "soft" rather than a strict "hard" for-loop, since a single latent step handles more than one exact hop. This soft-iteration hypothesis makes a further prediction: small perturbations to K should barely affect ***Bottleneck-latent***'s accuracy. Indeed, only ***Bottleneck-latent*** retains its performance under $K = 5, 6, 7$, whereas ***Full-latent*** is sensitive to $K = 5$ and ***Pause-token*** to $K = 7$ (Fig. 4C). Together, these results support that ***Bottleneck-latent*** may implement a "soft" forward search through its recurrent circuit.

3.3.2 Localize the sparse circuit inside the bottleneck latent model

Motivated by the RSA and intervention analyses above, we examine which components support the recurrent computation in ***Bottleneck-latent*** and how this circuit enables OOD generalization (Fig. 5) by circuit pruning to the recurrent updates from z_1 to z_K. We check all 20 physical components in ***Bottleneck-latent***, including $4\times$ attention heads and $1\times$ MLP per layer in GPTNeoX. A component is removed if the remaining circuit retains at least 90% output consistency with the full model and an $R^2 > 0.8$ for the candidate logit margin. This procedure (see A.6) greedily continues until no further component can be removed (Fig. 5A), leading to an 8-component sparse circuit (Fig. 5B) that preserves 91.9% output consistency and an R^2 of 0.837.

We next run the pruned circuit on other 7 to 12-hop samples that are not involved in pruning. The circuit retains 90.9% output consistency with the full model, above the 52.7% when these eight components are removed and 55.8% when a random size-matched subset is retained instead (Fig. 5C).

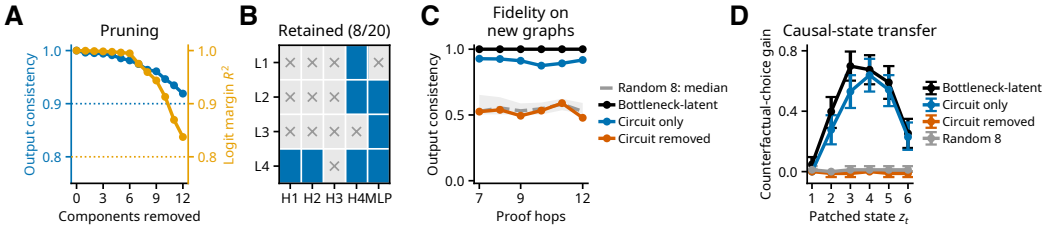

Figure 5: **Localization of the recurrent circuit in *Bottleneck-latent*.** Replacement based pruning (**A**) retains eight of twenty recurrent components (**B**). The selected circuit largely preserves candidate choices (**C**) and causal state-transfer effects (**D**), whereas removing it or retaining random size-matched components does not. Error bars indicate pointwise 95% bootstrap confidence intervals over base graphs; the gray band shows the 10th–90th percentiles across twenty random circuits.

To test whether the selected circuit preserves the causal recurrent mechanism in ***Bottleneck-latent***, we repeat the latent-state transplantation. On separate 8-hop pairs with a connectivity swap at depth 4, we measure the increase in counterfactual-answer choices relative to each condition's own baseline. The selected circuit retains a similar step-dependent transfer profile of the full model, with a peak increase of 63.9% versus 69.9%. In contrast, this effect is largely absent when the circuit is replaced by a random size-matched subset (Fig. 5D). In addition, the results remain stable across 7-12 hops (Fig. A.3). Together, our selected circuit preserves not only the model's output, but also the causal state-transfer mechanism identified above.

3.3.3 A recurrent search algorithm inside the sparse circuit

With the pruned circuit narrowing the recurrent computation to a 8 components, we next examine the specific role of each during recurrent computation. Among them, two components are particularly interesting. Attention head 1 in layer 4 (L4H1) separates how premise sources and destinations are transmitted. The corresponding MLP (L4MLP) helps propagate the retrieved information into subsequent recurrent states.

For a premise `A is B.`, A and B are referred to as the left-hand side (LHS) and the right-hand side (RHS), respectively. Using the same constructions above, we create the same candidate-switching interventions by either swapping the LHS or the RHS at the same premise location (Fig. 6A). By

replacing the actual cache with the swapped one, this matched-pair swap isolates whether the influence propagates through the RHS K-cache or the V-cache. We first characterize how attention reads graph premises. We find that, in L4H1, LHS swaps affect the answer mainly through keys, whereas RHS swaps affect mainly through values (Fig. 6B). To check if this division persists across different steps, we measure the similarity between the attention distribution before and after transplant using Jensen-Shannon distance, and find that these distributions are highly consistent (≈ 1, Fig. 6C). These results suggest an 'address–content' organization in L4H1: keys determine which nodes are linked, while values supply the destination node information.

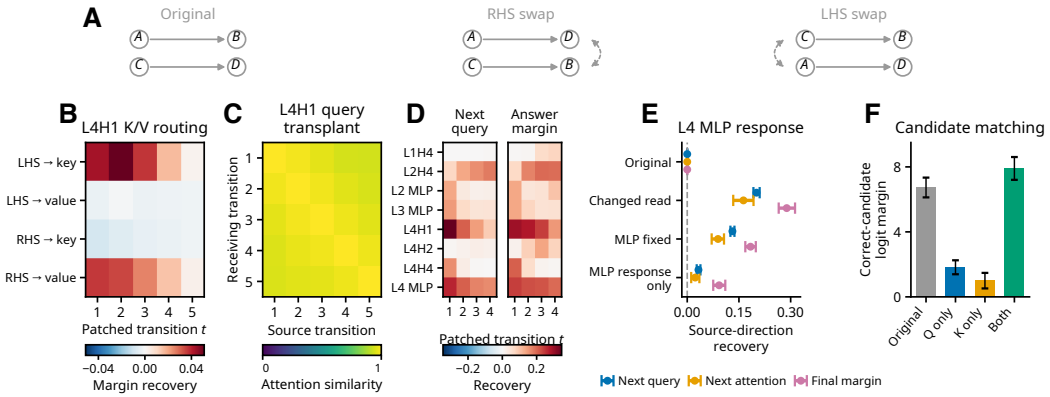

Figure 6: **Functional analysis of the recurrent circuit in *Bottleneck-latent*. A** illustrates the LHS and RHS swaps. Interventions within the pruned circuit distinguish L4H1's key/value routing (**B**), query-dependent selection of reading depth (**C**), and component contributions to the next query and final answer (**D**). Fixing or transplanting the L4 MLP response (**E**) separates its contribution from the residual pathway. **F** tests candidate matching through Q/K interventions in L4H1/H2/H4. Error bars indicate pointwise 95% bootstrap confidence intervals.

Then, we examine how MLP layers contribute to the computation. Instead of changing the structure of G, we swap the query root node r in matched chains used by Fig. 4, and measure how each component shifts the next query and the final answer in each transition (Fig. 6D). The L4MLP along with L4H1 show persistent influence on both, suggesting that they work together and change the query direction to further influence the next recurrent step. To isolate the contribution of L4MLP, we select the recurrent update $z_2 \to z_3$, use the swapped L4H1 and measure how L4MLP influences the downstream targets from next query to final answers under different interventions. With L4H1's output changed, we find recomputing L4MLP shifts all downstream targets to the swapped direction, compared with a frozen L4MLP (Fig. 6E). However, this effect is not additive (single L4MLP change barely shifts the direction) and relies on the residual stream (shifts exist even the MLP is fixed, though the magnitude is much lower). This shows that, L4MLP, they works by the whole residual stream, act as a "filter", to select information for the next round's operation.

We next ask how the evolving z_t becomes evidence for answer candidate c^*, given an r. Candidate positions in x carry graph conditioned representations that recurrent attention can read (during reasoning phase). Using the sparse pruned circuit above, we keep the original question intact and test two types of changes with L4H1/H2/H4 during $z_t \to z_{t+1}$. In the first change case, we replace the query with the Q from same step's update of a separate run starting from the other chain's root. For example, if the original graph contains $r \rightsquigarrow A$ and $s \rightsquigarrow B$, we replace $Q_{r,t}$ with $Q_{s,t}$, while the original question still asks about r. In the second condition, we use candidate keys obtained from a graph with exchanged candidate endpoints: $r \rightsquigarrow B$ and $s \rightsquigarrow A$, while keeping the question and its candidate positions unchanged. Either change alone reduce the c^* logit margin, while applying both changes together can restore it. This suggests the candidate evidence depends on a match between the current recurrent state and the candidates' graph context. Among the retained L4 Hs, H1 shows the strongest recurrent Q/K matching effects, whereas H2 shows the largest accuracy loss under candidate value exchange and the greatest logit margin recovery at final readout phase.

Together, these results reveal how a recurrent search algorithm is implemented in the ***Bottleneck-latent***: the z_t maintains currently reachable node in G; attention head L4H1 works as a soft tracer

and uses premise LHSs to retrieve their RHSs to expand the reachable set; and L4 MLP, together with the residual pathway, then incorporates the retrived information back to z_{t+1}, guiding the next round of seaching. The candidate reading pathways in L4H1/H2/H4 also connect this evolving state to answer evidence. The Q/K matching controls the candidate information written into the latent z_t, from which the final answer is decoded. These processes can operate in parallel, which allows the ***Bottleneck-latent*** to build a faster reachability search than strict step-by-step traversal as we have seen in 7 to 12 hop problems.

4 CONCLUSION

We ask whether models under different forms of thinking develop mechanistically distinct solutions, or converge on the same solutions through different ways. We train five GPT-like variants with the same backbone on an extended ProsQA task, and compare the mechanisms they induce.

Similar in-distribution performance hides the mechanistic divergence among the models. ***Direct***, ***Chain-of-Thought***, and ***Pause-token*** models solve in-distribution problems well, but rely on shortcuts related to local graph structure and generalize poorly to out-of-distribution problems. Notably, ***Chain-of-Thought***, which is explicitly supervised with step-by-step proofs, fails to generalize, suggesting that training on reasoning traces does not guarantee that a model will reason in the same way. In contrast, the latent models (***Full-latent*** and ***Bottleneck-latent***), despite receiving no intermediate reasoning trace or reward signal, develop a recurrent circuit that implements "soft" forward search algorithm, expanding the reachable set across recurrent steps and generalizing to problems with longer reasoning depths. Digging deeper into the circuit in the bottleneck model, we find that an attention head retrieves graph relations through an address–content organization, while an MLP, together with the residual stream, integrates the retrieved information into the next state and a multi-head reading mechanism performs the candidate matching.

Together, these results show that different thinking interfaces lead to distinct underlying mechanisms, even at similar performance. The latent-reasoning models that allow information to flow fully across steps develop genuine reasoning computation that matches the structure of the task.

5 DISCUSSION

We want to emphasize the importance of testing model behavior on OOD problems before turning to internal mechanisms. As the Stroop's color–word interference task reveals how humans process language and visual information, carefully designed OOD problems can also reveal how a model solves a task and guide where further mechanistic analysis should go (Friedman et al., 2024).

Building on this behavioral comparison, our design also isolates the effect of the thinking interface itself. Most mechanistic studies analyze a single model or a single form of reasoning in isolation. Instead, we train five variants that share the same backbone, dataset, and budget and differ only in their thinking interface, separating the mechanistic differences attributed to the interface from others. One family that is commonly used but not included here is the looped transformer, which applies the same weight-tied block for multiple iterations (Dehghani et al., 2019; Giannou et al., 2023). This explicit recurrence formalizes the iterative computation that we discover in the latent models, but whether the latent models generate the same forward search is still left to future work.

Finally, our conclusions come from small models on a controlled symbolic reasoning task, which simplifies analysis and lets us narrow the computation down to an interpretable circuit with confidence. However, the use of small models and a synthetic task limits how far the current conclusions can extend. Thus, whether our results and conclusions hold for larger pretrained LLMs and more naturalistic problems remains an open question for future work.

AI USE STATEMENT

In this work, we used generative AI tools to fix grammar issues and improve the clarity and readability of the manuscript text.We did not use generative AI tools for ideation, experimental design, methodology, data analysis, or the drafting of any original scientific content. All AI-assisted edits

were reviewed and verified by the authors. The authors take full responsibility for the final content of this work, including all text, claims, figures, and artifacts produced with generative AI.

Ethics statement

This work does not involve any human subjects, human-derived data, or deployed systems. We do not foresee ethical concerns such as privacy, bias, or safety risks arising from this work.

Reproducibility statement

Our code is available at `https://anonymous.4open.science/r/Not-All-Thinking-Is-Created-Equal-E00B`. Dataset generation and training details are provided in Appendices A.1 and A.2, respectively.

References

Darpan Aswal, Thomas Palmeira Ferraz, Yongxin Zhou, and Maxime Peyrard. Observable patterns are not explanations: A causal-geometric analysis of latent reasoning models, June 2026.

Jannik Brinkmann, Abhay Sheshadri, Victor Levoso, Paul Swoboda, and Christian Bartelt. A mechanistic analysis of a transformer trained on a symbolic multi-step reasoning task. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4082–4102. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.242. URL https://aclanthology.org/2024.findings-acl.242/.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024. URL https://jmlr.org/papers/v25/23-0870.html.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Dan Friedman, Andrew Kyle Lampinen, Lucas Dixon, Danqi Chen, and Asma Ghandeharioun. Interpretability illusions in the generalization of simplified models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*. PMLR, 2024. URL https://proceedings.mlr.press/v235/friedman24a.html.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens, 2024.

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, November 2025.

Namgyu Ho, Laura Schmid, and Se-Young Yun. Large language models are reasoning teachers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14852–14882. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.830. URL https://aclanthology.org/2023.acl-long.830/.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2022.

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, July 2023.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason, 2023. URL https://arxiv.org/abs/2212.08410.

Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models, 2024.

Michael Rizvi-Martel, Guillaume Rabusseau, and Marius Mosbach. The illusion of superposition? a principled analysis of latent thinking in language models, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. SIM-CoT: Supervised Implicit Chain-of-Thought, 2025.

Yiwei Wu, Atticus Geiger, and Raphaël Millière. How do transformers learn variable binding in symbolic programs? In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 67284–67299. PMLR, 2025. URL https://proceedings.mlr.press/v267/wu25j.html.

Jiachen Zhao, Yiyou Sun, Weiyan Shi, and Dawn Song. Can aha moments be fake? towards quantifying decorative and true thinking in chain-of-thought, 2026.

Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought, 2025.

A Appendix

A.1 Dataset Generation algorithm

We construct ProsQA-Ext from the graph-reachability task in ProsQA (Hao et al., 2025). In ProsQA, each sample consists of a series of premises describe a DAG and a question asks which of the two candaites is reachable from the specified root node. In ProsQA-Ext, we preserved this logic, while we changed the how each sample is constructed. Labels are assigned after graph construction, premises are shuffled, and the correct candidate appears equally often on either side of the question, so that no obvious superficial statistics are related to the answer.

For the extended 400k 3–6 hop training set, we sample new graphs with an empirical quota that keeps the empirical joint distribution of graph size, proof length, queried root, and binned number of shortest paths similar to ProsQA. For ID validation and test, we retain the released graphs and queries, but reassign node labels and randomize premise and candidate order.

For OOD 7–12 hop dataset for evaluation, we use algorithm 1 to generate it:

Algorithm 1 Controlled long-hop graph generation

Require: Proof length $H \in \{7, \ldots, 12\}$; balanced candidate-side assignment b

Construct two vertex-disjoint directed paths P_1, P_2, each of length H
for $j \in \{1, 2\}$ **do**
    Add $13 - H$ off-path nodes U_j
    Attach the first node of U_j to the root of P_j
    Attach each remaining node of U_j to an internal node of P_j or an earlier node of U_j
end for
Sample additional edges into off-path nodes, without duplicates or cycles, until $|E| = 38$
Choose one of P_1, P_2 as the queried component
Set r to its root, c^+ to its endpoint, and c^- to the endpoint of the other path
Randomly assign node labels and permute the premise order
Place c^+ on side b of the binary question

A.2 Training details

All five variants are trained from random initialization on the same 400k ProsQA-Ext problems with proof lengths of 3–6 hops, separately on two Nvidia 5090 and 4090 machines. Each model uses a four-layer GPTNeoX backbone with hidden size 256, four attention heads, feed-forward dimension 768, untied input and output embeddings, and attention and hidden dropout of 0.1. We use AdamW with a constant learning rate of 4×10^{-4}, $(\beta_1, \beta_2) = (0.95, 0.999)$, weight decay of 10^{-4} and a batch size of 256. Training used BF16 without learning-rate warmup or decay. The objective averages next-token cross-entropy over prompt and answer tokens with equal per-token weights; Text CoT additionally supervised the shortest-proof tokens. Pause and latent variants used $K = 6$ thinking positions. Latent models are optimized end-to-end through the full recurrence without discrete intermediate targets, curriculum training or RL.

A.3 Observed correlation between models' internal computation and the algorithm intermediate variables

We compare model representations with propagated frontiers under forward search and successive ancestor sets of the candidates under backward tracing. For a graph $G = (V, E)$ with query root r, the forward frontier is initialized at the root and updated by following outgoing edges:

$$F_0 = \{r\}, \qquad F_{d+1} = \{v \in V : \exists u \in F_d,\ (u, v) \in E\}.$$

Each update expands all nodes in the current frontier, allowing multiple branches to be followed in parallel. Thus, F_d contains nodes reachable from r by a directed path of exactly d edges. It differs from the cumulative reachable set $\bigcup_{j=0}^{d} F_j$, which retains nodes reached at earlier depths. A node can appear in multiple frontiers if paths of different lengths lead to it. For example, edges $r \to a$, $r \to b$, and $a \to b$ give $F_1 = \{a, b\}$ and $F_2 = \{b\}$.

For backward tracing, we start from both candidates simultaneously and propagate their joint frontier along incoming edges:

$$U_0 = \{c_0, c_1\}, \qquad U_{d+1} = \{u \in V : \exists v \in U_d,\ (u, v) \in E\}.$$

Here, U_d is the union of both candidates' frontiers after exactly d reverse steps. For RSA, forward and backward frontiers are encoded as binary vectors over node labels, indicating membership in the set. The backward representation is invariant to the order of candidates in the query.

The analysis covers the final prompt position and the thinking phase, where applicable, on both ID and OOD problems. For latent models, we use the continuous states z_t, with z_1 defined as the normalized last-layer state at the final prompt position. For ***Chain-of-Thought***, we use the first 21 generated proof tokens, matching the shortest generated proof among the samples. For ***Pause-token***, we analyze the normalized last-layer residual states $N_\theta(h_{n+t}^{(L)})$, since its input vectors z_t are identical across steps. ***Direct*** serves as a control without additional thinking steps, using normalized last-layer states at the final prompt position and during answer readout. Figure A.1 shows forward-propagation alignment across all evaluated depths.

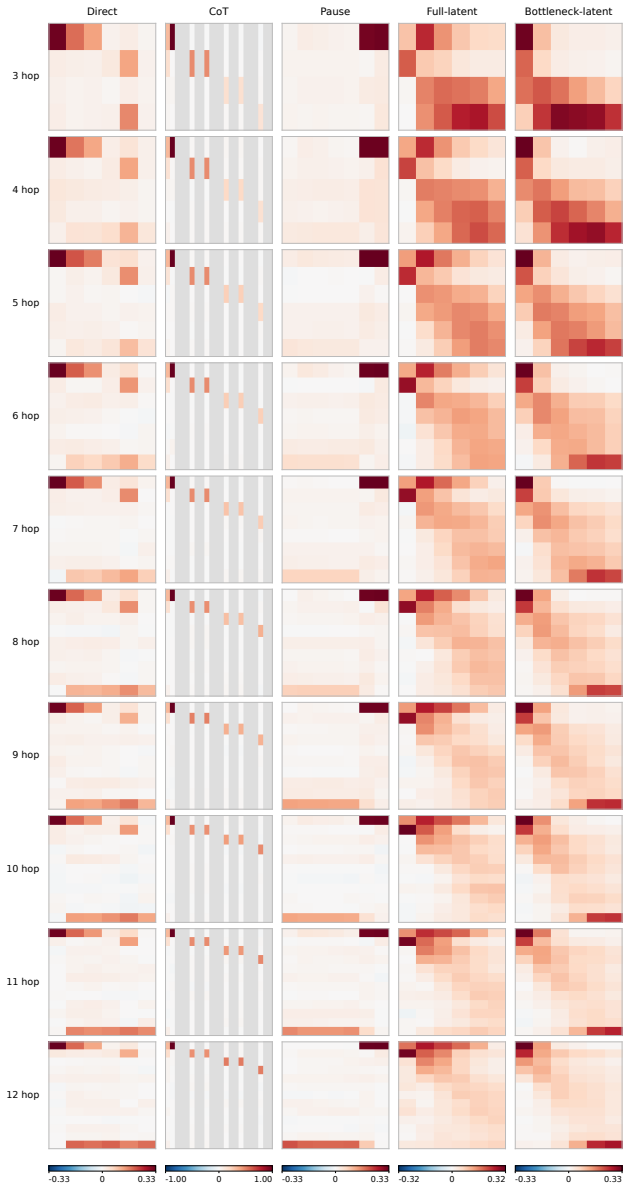

Figure A.1: **Representational alignment with forward graph propagation across task depths.** RSA heatmaps for five model variants on 3 to 12-hop problems (rows), across ID (3–6 hops) and OOD (7–12 hops) conditions. Entries represent Spearman correlations between model-representation and frontier dissimilarities, as shown in Fig. 2.

A.4 DIAGONALITY

We develop *Diagonality* to quantify whether stronger alignment shifts toward later computation positions as algorithm depth increases. Let $R_d(t)$ denote the RSA value at depth d and position t. First, we convert each row to normalized ranks, $r_d(t) = (\mathrm{rank}(R_d(t)) - 1)/(n_d - 1)$, where n_d counts finite entries. Undefined entries and rows with fewer than two finite entries are set to zero.

We the compute the best fixed position, independently selected row maxima, and the best nondecreasing path:

$$S_{\text{static}} = \max_t \sum_d r_d(t), \qquad S_{\text{top}} = \sum_d \max_t r_d(t), \qquad S_{\text{fwd}} = \max_{t_0 \le \cdots \le t_D} \sum_{d=0}^{D} r_d(t_d). \tag{2}$$

We define *diagonality* as

$$\text{Diagonality} = \frac{S_{\text{fwd}} - S_{\text{static}}}{S_{\text{top}} - S_{\text{static}}}. \tag{3}$$

The score lies in $[0, 1]$: when the denominator is positive, one means that a nondecreasing path reaches every row's maximum (perfect propagation), while zero means that allowing forward propagation gives no advantage over a fixed position. We assign zero when the denominator vanishes.

Note that *Diagonality* measures the ordering of alignment, not its absolute strength. It allows pauses and jumps between positions, without requiring one model step per algorithm update.

Across numer of hops, no variant shows a clear, consistent pattern of alignment with backward search (Fig. A.2).

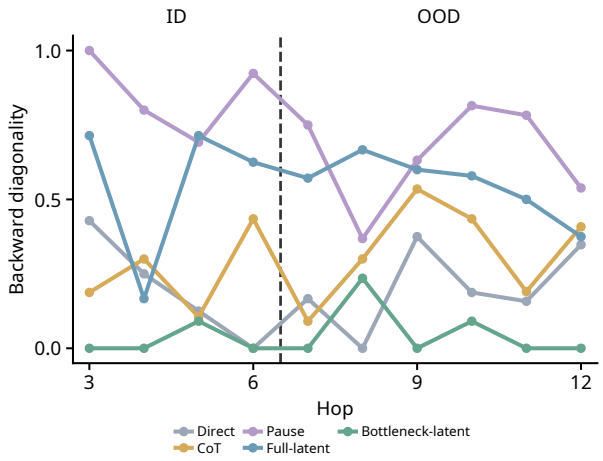

Figure A.2: **Diagonality of alignment with parallel backward search.** The same diagonality measure is applied to RSA against the joint backward frontier for all five model variants. The dashed line separates ID from OOD problems.

A.5 Effect of hop perturbation on pruned circuit in Bottleneck-latent model

Figure A.3 shows the effect of hop perturbation on state transfer in the pruned circuit across 7–12-hop problems.

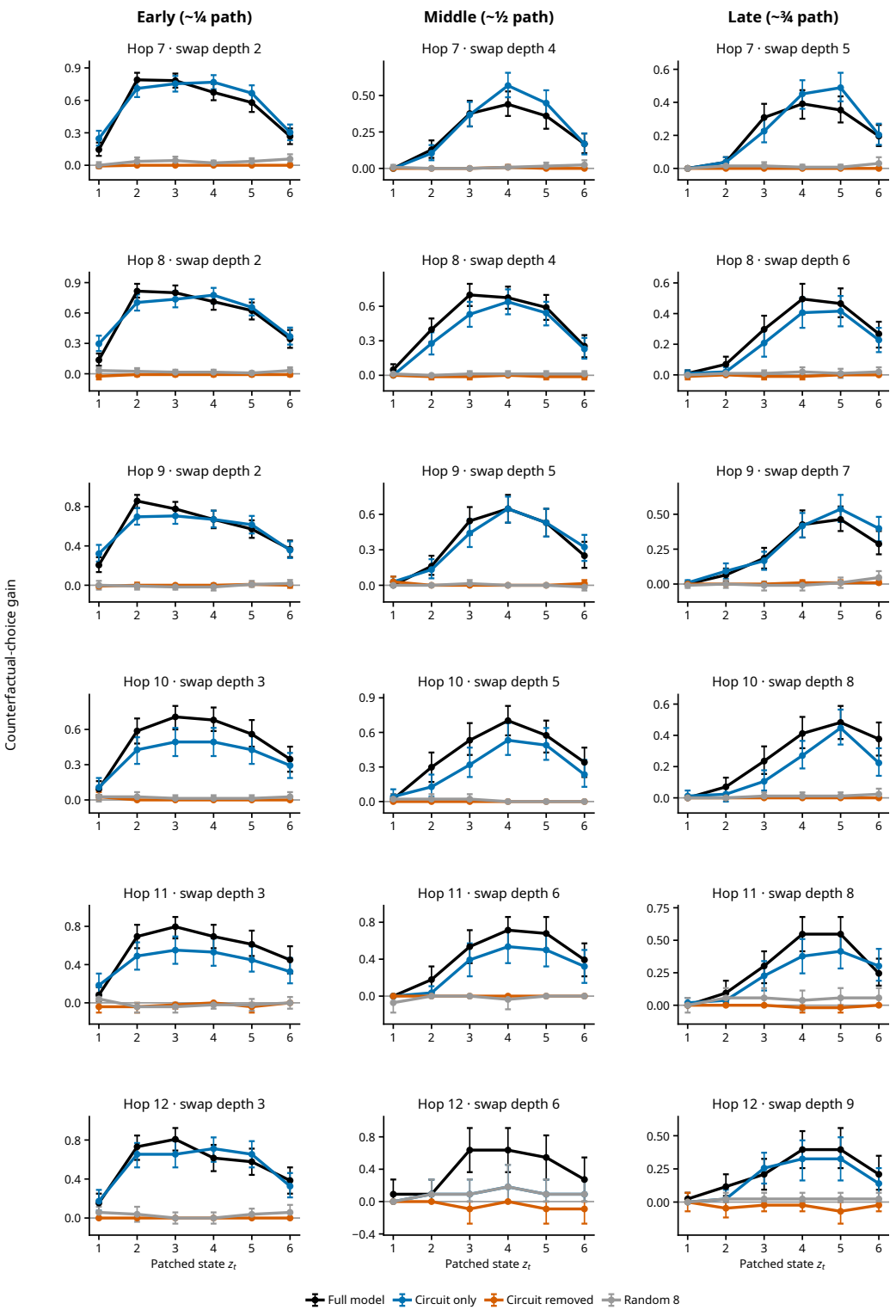

Figure A.3: **Effect of hop perturbation on pruned circuit in Bottleneck-latent model.**

A.6 RECURRENT CIRCUIT PRUNING

We apply greedy mean-replacement pruning (Algorithm 2) to the 20 recurrent components of ***Bottleneck-latent*** (16 attention heads and four MLPs). Each component is pruned across all five recurrent transitions from z_1 to z_6. Replacement means are computed separately for each component and transition from both members of counterfactual pairs constructed from 512 independent 8-hop graphs. These means remain fixed throughout pruning. Input encoding and answer readout remain the same.

For a retained component set S, let $m_i(S)$ denote the candidate logit margin after mean-replacing all components outside S, and let m_i denote the full-model margin. We measure candidate-choice consistency $C(S)$ and margin fidelity $R^2(S)$ on the examples:

$$C(S) = \frac{1}{N}\sum_i \mathbf{1}[\mathbf{1}[m_i(S) \geq 0] = \mathbf{1}[m_i \geq 0]], \qquad R^2(S) = 1 - \frac{\sum_i (m_i(S) - m_i)^2}{\sum_i (m_i - \bar{m})^2}.$$

Algorithm 2 Greedy recurrent circuit pruning

Require: Full component set $\mathcal{C}$; fixed replacement means

$S \leftarrow \mathcal{C}$
while $S \neq \varnothing$ **do**
  for each $c \in S$ **do**
    Evaluate $C_c = C(S \setminus \{c\})$ and $R_c^2 = R^2(S \setminus \{c\})$
  end for
  $\mathcal{F} \leftarrow \{c \in S : C_c \geq 0.90,\ R_c^2 \geq 0.80\}$
  if $\mathcal{F} = \varnothing$ **then**
    break
  end if
  $c^\star \leftarrow \arg\min_{c \in \mathcal{F}} \frac{1}{2}\left(\frac{1 - C_c}{0.10} + \frac{1 - R_c^2}{0.20}\right)$
  $S \leftarrow S \setminus \{c^\star\}$
end while
return S

A.7 HEAD CONTRIBUTIONS DURING RECURRENCE AND ANSWER READOUT

We compare L4's H1, H2, and H4 using the same ***Bottleneck-latent*** recurrent circuit (Fig 5).

During recurrence, we apply the Q/K interventions in Fig. 6F to one head at a time across all five updates. We measure the decrease in the $correct - incorrect$ candidate logit margin. We also test candidate values separately: we reverse the candidate order in a separate run and use its candidate position values in the original run, keeping the attention weights unchanged at each update. We measure the resulting accuracy loss in percentage points.

For final answer readout, we use graph pairs with opposite correct answers but the same task root, candidate positions, and answer prefix. We replace one head's output at the answer prediction position with its output from the opposite-answer run. The original latent trajectory stays fixed. 43 of 96 graph pairs for which both answers are initially predicted correctly are used for analysis. Here, the margin is the right candidate's logit - the remaining candidate's logit. Recovery measures the shift toward the opposite answer margin: 0% means no change, and 100% means reaching that margin.

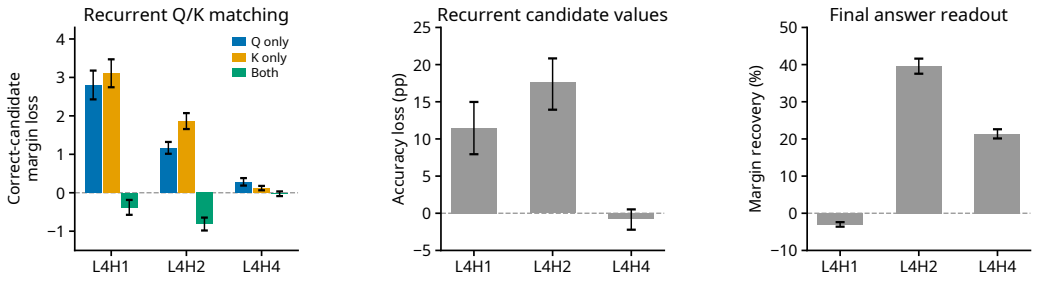

Figure A.4: **Head contributions during recurrence and answer readout.** Individual interventions compare recurrent Q/K matching (left), candidate-value exchange (middle), and final answer readout (right) for L4H1, L4H2, and L4H4. Error bars indicate pointwise 95% bootstrap confidence intervals.