How Large Reasoning Models Solve Problems: A Mechanistic Study on Tower of Hanoi

Yukang Yang [1] **Mengdi Wang** [1] **Jonathan Cohen** [2] **Taylor Webb** [3]

Abstract

Large reasoning models (LRMs) have exhibited strong reasoning abilities widely attributed to deliberate thinking through long internal chains of thought (CoT). However, there has been significant debate about whether the content of CoT traces is causally relevant for the performance of these models. In this paper, we conduct a mechanistic study of CoT traces on a task adapted from the classic Tower of Hanoi puzzle to understand how open-source LRMs solve problems. We selected gpt-oss 20b and gpt-oss 120b, both of which achieve around 95% accuracy on the task under high reasoning effort. We identified 6 dominant reasoning patterns deployed for finding solutions, including analogy identification, human-like strategic problem solving, and systematic search. Focusing on strategic problem solving episodes, we identified self-verification as an important reasoning primitive guiding the outcome of these episodes. By intervening on the verification process — either correcting verification errors or corrupting correct verifications — we established a causal link between these self-verification behaviors and the accuracy of the final solution. Using causal mediation analysis, we further identified a sparse set of experts in the Mixture-of-Experts layers with strong causal effects on verification behavior, and showed that the outputs of these experts can be aggregated to induce verification behavior. Together, these experiments provide causal evidence that the content of CoT traces, and self-verification behaviors in particular, meaningfully shape the final answers generated by LRMs, and begin to uncover the internal mechanisms that support these capacities.

[1]Department of Electrical and Computer Engineering, Princeton University, NJ [2]Princeton Neuroscience Institute, Princeton University, NJ [3]Mila – Québec AI Institute, Université de Montréal. Correspondence to: Taylor Webb <taylor.w.webb@gmail.com>.

Mechanistic Interpretability Workshop at the 43rd International Conference on Machine Learning, Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

1. Introduction

Chain-of-thought (CoT) reasoning has become a critical component of the state-of-the-art large language models (LLMs), empowering advanced reasoning, planning, and problem solving (Jaech et al., 2024). Large reasoning models (LRMs) are a specialized class of LLMs, trained largely via reinforcement learning to perform internal CoT reasoning before producing a final response (Guo et al., 2025; Jaech et al., 2024). Currently, most state-of-the-art LLMs are either dedicated reasoning models (e.g., OpenAI o3 (OpenAI, 2025) and DeepSeek R1 (Guo et al., 2025)) or hybrid models that support both CoT thinking and instant responses (e.g., Gemini 3.1 Pro (Google DeepMind, 2026) and Claude Opus 4.7 (Anthropic, 2026)).

Although LRMs have achieved strong reasoning capabilities through long CoT traces, it remains unclear whether the content of these traces is causally relevant for model performance (Kambhampati et al., 2025). Several works (Turpin et al., 2023; Chen et al., 2025; Arcuschin et al., 2025; Zhao et al., 2025) have questioned the faithfulness of CoT traces in verbalizing the true reasoning processes that drive a model's final responses, or have identified unreliable correlations between reasoning traces and the final solutions generated by these models. On the other hand, specific cognitive behaviors such as verification and backtracking have been shown to be a critical prerequisite for the success of reinforcement learning for reasoning models (Gandhi et al., 2025), suggesting that the content of such reasoning behaviors may indeed be important for model performance.

To shed light on these questions, and gain a systematic understanding of how large reasoning models solve problems, we adapted the classic Tower of Hanoi (ToH) puzzle and analyzed the CoT traces produced by two high-performing open-source RL-finetuned reasoning models, gpt-oss 20b and gpt-oss 120b. We analyzed these traces at three levels of granularity. First, we identified, and quantitatively characterized, the high-level reasoning strategies that these models use to solve problems, including diverse strategies such as strategic problem solving, formal search, and analogical reasoning. Second, focusing on strategic problem solving, we identified the functional components that these models used to carry out this particular reasoning strategy, includ-

ing subgoal proposal, action proposal, and verification. We tested the causal relevance of a particular reasoning component, verification, by intervening at the token level, finding that this behavior was critical for the success of reasoning – incorrect solutions could be remedied by correcting verification errors, and corruption of correct verifications led to incorrect solutions. Finally, we investigated the internal mechanisms involved in generating this behavior, identifying a sparse set of experts in the Mixture-of-Expert layers that play an especially important role in implementing verification. Taken together, these results provide evidence for a strong causal link between the content of CoT traces and the accuracy of the downstream solutions generated by LRMs, and shed light on the underlying mechanisms of problem-solving in these models.

2. Behavioral Analyses

We studied the Tower of Hanoi (ToH) task, a classic puzzle used to study multi-step problem solving in psychological research. The standard ToH task is well-defined under simple deterministic rules and clear goals and has well-known optimal solutions, which creates a verifiable and controllable testbed for analyzing LRM problem-solving behaviors. There is also a wealth of knowledge about how human problem solvers approach this task which can potentially be used to better understand the strategies employed by LRMs.

2.1. Task Design

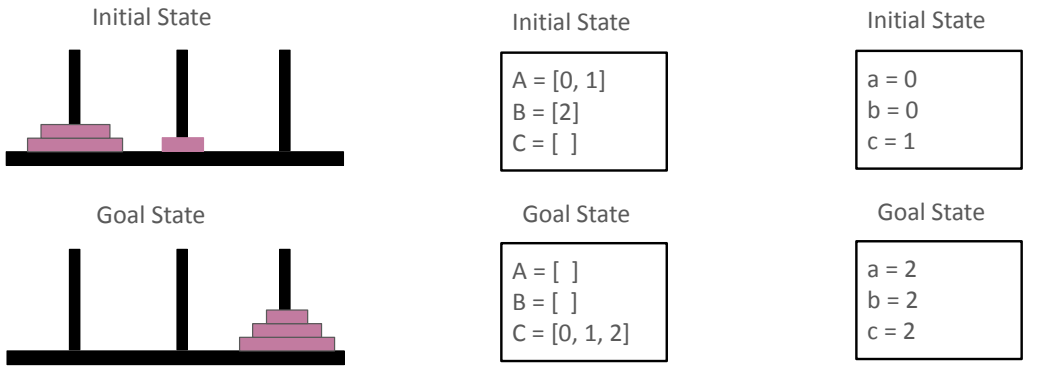
In a typical ToH setting, there are three pegs with n disks of different sizes (typically stacked on the first peg). The goal is to transfer all disks to a target configuration (typically all disks stacked on the third peg) by making valid moves according to a set of rules. To test text-only LRMs, and also to reduce the possibility that LRMs can solve the task by recalling standard solutions from their training data, we reformulated original problem to one involving lists and digits instead of pegs and disks, similar to the approach of Webb et al. (2025). We evaluated LRMs on problems with various initial configurations (rather than only the canonical starting configuration).

Inspired by classic work on human problem solving (Kotovsky et al., 1985), we adopted two isomorphic versions of this task, referred to as *Move* and *Change* problems. For *Move* problems, there are three lists labeled A, B, and C with n numbers distributed across these lists (Figure 1b). The goal is to *move* all numbers to list C, arranged in ascending order, subject to the two following rules: (1) you can only move a number if it is at the rightmost end of its current list, and (2) you can only move a a number to another list if it is larger than all other numbers in that list. For *Change* problems, there are n variables named in alphabetical order (a, b, c, ...) each assigned a value of 0, 1, or 2 (Figure 1c). The goal is to *change* all variables to be equal to 2, subject to the two following rules: (1) if two variables are assigned equal values, you can only change the value of the variable that comes later in the alphabet (e.g., if $a = 0$ and $c = 0$, you can only change c), and (2) when changing the value of a variable, you cannot change it to a value that has been assigned to a variable that comes later in the alphabet. We chose to investigate these problem variants because they have been shown to elicit different patterns of behavior in human psychology experiments (Kotovsky et al., 1985). We selected $n = 3$ for both types of problems, and tested on all $3^3 - 1 = 26$ possible initial configurations. Figure 1 shows examples of each type of problem and Figure 15 provides corresponding user prompts.

2.2. Model Performance

There are two dominant training paradigms used for training models to perform sequentially extended processing (i.e., CoT) before emitting a final answer. The first is fine-tuning with reinforcement learning from verifiable rewards (RLVR); the second is distilling a smaller student model from a stronger teacher model (typically trained with RLVR) such as DeepSeek-R1. We evaluated the performance of 17 leading open-source reasoning models across these two training paradigms. We tested models on 26 initial problem configurations with 5 runs per configuration. As shown in Figure 2, distilled models were consistently worse than models trained with RLVR. Among all models tested, we identified gpt-oss 20b, gpt-oss 120b (both with high reasoning effort), and Qwen3-235B-A22B-Thinking-2507 as high peforming models, all achieving around 95% accuracy on this task.

Comparing Performance in Move vs. Change Problems. We chose the 6 models with the best performance on move problems and evaluated their performance on change problems (Figure 3). A classic result from psychological studies is that human problem solvers take about twice as long to solve change problems as they take to solve move problems, even though these tasks are formally isomorphic, and even when accuracy is balanced between these two task variants (Kotovsky et al., 1985). We found that all 6 of the models that we tested displayed a qualitatively similar effect, producing about twice as many reasoning tokens to solve change vs. move problems, even for the three best performing models that displayed comparable accuracy on both tasks (Figure 3b). These results suggest that LRMs may employ similar strategies as human problem solvers. We investigate the specific strategies that LRMs use to solve these problems in the following section.



(a) Original Version *(b) Move* Problem *(c) Change* Problem

Figure 1. **Tower of Hanoi (ToH) Task Designs.** (a): the original version with physical pegs and disks; (b, c): two isomorphic text-based versions used in our study, *Move* and *Change* problems.

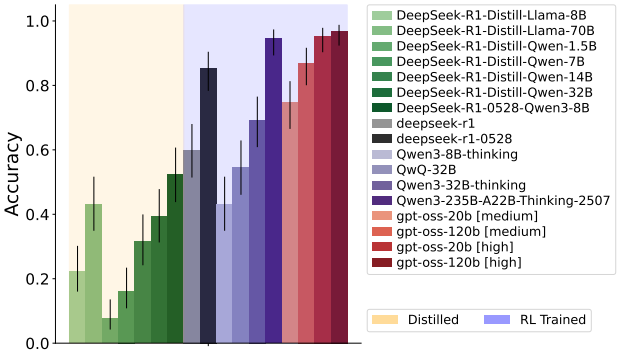
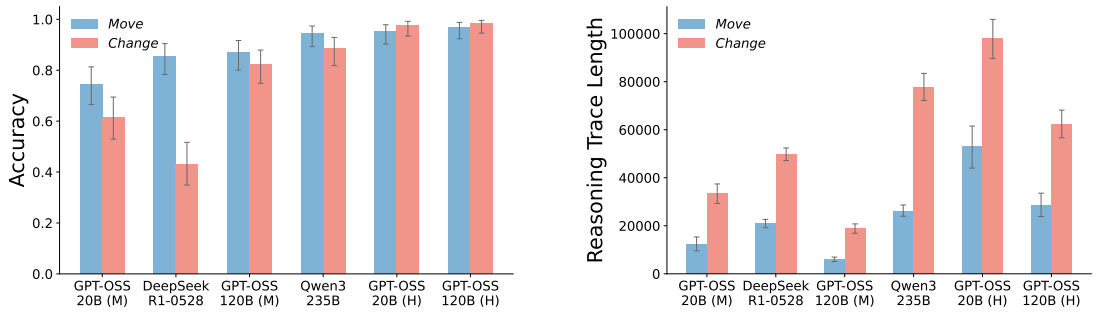


Figure 2. **Large Reasoning Model Performance on Move Problems.** Performance across 5 runs each for 26 ToH Move problems. Error bars reflect binomial confidence intervals.

2.3. Reasoning Strategies in Chain-of-thought Traces

To better understand the reasoning strategies employed by LRMs, we analyzed the traces generated by the gpt-oss models, which performed nearly perfectly with high reasoning effort, while solving ToH move problems. We discovered 6 dominant reasoning strategies in these models' CoT traces. These strategies were typically clustered within contiguous segments of tokens (referred to here as episodes), with multiple strategies being employed throughout a single trace.

- **Problem Understanding**: In these episodes, the model restated and interpreted the goal, formalized the task, and clarified the rules via rephrasing (not tied to a specific problem configuration). This behavior typically occurred early in a reasoning trace.
- **Analogy Identification**: In these episodes, the model identified potential analogies between the current task and similar puzzles or problems. The text for these episodes may include (1) identifying a similar puzzle (e.g., Tower of Hanoi) as a template, (2) comparing the current problem with the known template and mapping the entities and rules, (3) recalling known strategies, heuristics, or canonical solutions, and (4) adapting them to the current problem to construct a solution.
- **Strategic Problem Solving** : In these episodes, the model applied human-like problem-solving strategies, including means-ends analysis and trial-and-error to search for a solution in an informal, exploratory manner. The text for these episodes often involves analyses of task constraints and decomposition of goals into subgoals, as well as other behaviors such as action proposal, verification of proposed actions, and backtracking.
- **Systematic Search**: In these episodes, the model formalized the task as a state graph and systematically applied formal search algorithms such as breadth-first search (BFS) or depth-first search (DFS). Each node represents a specific task state.
- **Path Simulation**: In these episodes, the model simulated the entire path generated in a previous episode (e.g. a strategic problem-solving episode), verifying each move step by step, and confirming that the goal is reached after the final move.
- **Answer Packaging**: In these episodes, the model formatted its final answer in the required template (e.g., representing each move as "Step 1: Move N from list X to list Y").
- **Others**: This category includes sentences that do not clearly fall into any of the categories above, or consist only of meta-commentary without substantial reasoning content. This category was also used as a placeholder for any missing reasoning categories.



(a) Model Accuracy *(b)* Reasoning Trace Length

Figure 3. **Comparison Between Move and Change Problems.** (a): Accuracy on move vs. change problems. Error bars reflect binomial confidence intervals. (b): Length of reasoning traces for move vs. change problems. Error bars reflect bootstrap confidence intervals. Reasoning trace lengths are counted in characters.

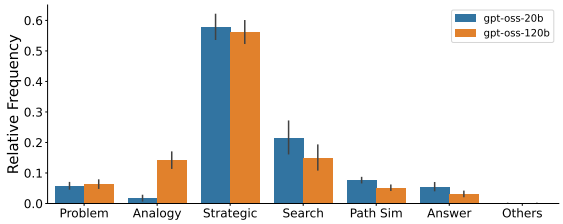


Figure 4. **Distribution of Reasoning Patterns.** Relative frequencies are computed at the character level. We selected the 16 most challenging problems, for which the optimal solution required at least 5 moves, and performed 5 runs per problem. Error bars reflect bootstrap confidence intervals.

We used Gemini-3.1 Pro as a parsing agent to classify the reasoning traces into the aforementioned reasoning patterns. To improve the reliability of this approach, we described the typical behaviors associated with each category in the system prompt and provided manual annotations as in-context examples. As illustrated in Figure 4, both models predominantly deployed human-like, strategic problem-solving strategies. The 120b model had a much higher tendency to identify analogies with existing puzzles such as the canonical ToH, despite the lack of any explicit hints about the relationship of the task to ToH. For problems that shared the canonical starting configuration with the original ToH (all disks stacked on the left peg, or all numbers arranged in list A), this strategy was sometimes successful (3 out of 10 runs, an example shown in Figure 14), as the model could recall the specific optimal 7-move solution for ToH and map it on to the text-based problem, but for other starting configurations this strategy was typically not effective. Both models tended to adopt systematic search as a fallback strategy: either to identify a candidate path after initial strategic problem-solving attempts failed, or to find the optimal path using formal methods such as BFS, as shown in Figure 9. The gpt-oss 20b model deployed formal search methods more frequently.

3. Components of Strategic Problem-Solving in LRMs

Our analysis of LRM reasoning traces indicated that strategic problem-solving was the dominant strategy used for solving ToH problems. To better understand this strategy, we further divided strategic problem-solving episodes into the following set of functional components:

- **Subgoal Proposal**: Given a goal and the current state, a subgoal is proposed, for example by (1) comparing the goal state with the current state and proposing intermediate goals that gradually shrink the gap (i.e., means-ends analysis), or (2) identifying obstacles that block an attempted action and establishing the subgoal of removing them.
- **Action Proposal**: A specific action is proposed, for example by random exploration or by grounding an under-specified goal into a concrete action (e.g., specifying the destination list for the goal "move 1 out of C").
- **Verification**: A proposed action is evaluated against the rules of the task, determining whether the action is valid.
- **Action Execution**: A proposed (and potentially verified) action is added to the current plan.
- **Backtracking**: Recently taken actions are abandoned, and the task state is reverted to a previously visited configuration (typically when exploration reaches a dead end).

To gain a better understanding of the causal role that these components play in supporting the reasoning capabilities of LRMs, and the internal mechanisms that are responsible for implementing these components, we chose to focus on the

verification component as a case study. Verification is well-defined in our task, as it involves evaluating the validity of proposed actions given the task rules and current state. Additionally, the ability of LRMs to self-verify, and the causal relevance of these behaviors for reasoning performance, has been a major locus of debate (Stechly et al., 2024; Guo et al., 2025; Zhao et al., 2025). To shed light on these issues, we first performed experiments in which we intervened on verification events at the token level, either correcting incorrect verifications or corrupting correct ones, and quantified the effect on downstream performance (Section 3.1). Additionally, we investigated the internal mechanisms that control the generation of this behavior (Section 3.2).

3.1. Reasoning Trace Intervention

We first investigated the correlation between the correctness of verification events and the correctness of the final solutions generated by the model. We used the same 80 runs (16 problems $\times$ 5 runs each) of challenging problems that were used to investigate the distribution of reasoning patterns (in Figure 4). For each strategic problem-solving episode parsed in Section 2.3, we deployed Gemini-3.1 Pro as a parsing agent to identify the moments where a path was discovered, i.e., where the model identified that the goal had been reached, and segmented each episode into distinct path-exploration segments accordingly, using Gemini-2.5 Pro as a reviewing agent to verify this segmentation. We selected all runs that contained at least one complete path-exploration segment, and we forced the model to generate its final answer after the *first* segment was complete. When evaluating the performance following only these initial reasoning segments, the accuracies of gpt-oss 20b and gpt-oss 120b dropped to 57.1% (40/70) and 82.5% (52/63), respectively, compared with 94.3% (66/70) and 95.2% (60/63) following the full reasoning traces.

Correlations Between Verification and Solution Accuracy. We identified all verification errors produced during the first complete path-exploration segments using Gemini-2.5 Pro as a parsing agent. A verification error refers to an incorrect judgment about the validity of the move to be verified, i.e., characterizing an invalid move as valid or vice versa. There was a clear relationship between whether the model got a particular problem correct and whether the reasoning segment contained at least one verification error (gpt-oss 20b, $\phi = 0.678$, $p = 1.4 \times 10^{-8}$; gpt-oss 120b, $\phi = 0.380$, $p = 0.0026$), as illustrated in Table 1.

Verification Intervention. We further examined the causal impact of verification on the solutions generated by the model by manipulating the verification process in two ways:

(1) Error Correction. To investigate the impact of correcting the model's verification errors, we first identified all path-exploration segments that contained at least one verification error, and for which the subsequent solution generated by the model was incorrect. We iteratively corrected these verification errors: correcting the first error, then letting the model continue to generate the reasoning traces for a fixed number of tokens, then identifying whether this continuation contained any new verification errors, and so on, until the model generated a complete path. This procedure was implemented by three separate agents (all based on Gemini-2.5-Pro) for detecting errors, revising errors, and identifying when a complete path had been generated. Figure 13a shows an example of this process. This procedure substantially improve the accuracy of the final solutions generated by the models, correcting 100% and 71% of the incorrect solutions generated by gpt-oss 20b and 120b respectively (Table 1).

(2) Error Injection. We also investigated whether injection of verification errors would cause the models to generate incorrect solutions. We identified all path-exploration segments that contained no verification errors, and for which the subsequent solution generated by the model was correct. We iteratively corrupted all verification events in these segments (using a similar methodology as the one employed for error correction) such that all moves were declared to be valid regardless of ground-truth validity. This procedure substantially impaired the accuracy of the final solutions generated by the models, resulting in 97% and 83% error rates for previously correct solutions generated by gpt-oss 20b and 120b respectively (Table 1).

These results demonstrate that verification behaviors have a profoundly important causal role in shaping the solutions that LRMs generate on this task. We next sought to understand the internal mechanisms that these models use to deploy this behavior.

3.2. Mechanistic Interpretability

Going beyond intervention at the token level, we next investigated the causal relevance of internal model activations in initiating verification behaviors.

Causal Mediation Analyses. We performed causal mediation analysis (CMA) (Meng et al., 2022; Todd et al., 2023) to estimate the causal effects of output embeddings for specific model components (e.g., attention heads or MLP layers). Focusing on the same reasoning segments investigated in the previous section, we first identified the token positions corresponding to the start of each verification event. We

	Answer correct	Answer incorrect
0 Verification errors	35	6
$\geq$ 1 Verification errors	5	24
Total	40	30

(a) gpt-oss 20b

Intervention type	Answer flip rate
Error correction	24 / 24 (100%)
Error injection	34 / 35 (97%)

	Answer correct	Answer incorrect
0 Verification errors	42	4
$\geq$ 1 Verification errors	10	7
Total	52	11

(b) gpt-oss 120b

Intervention type	Answer flip rate
Error correction	5 / 7 (71%)
Error injection	35 / 42 (83%)

Table 1. **Relationship Between Verification and Accuracy**. Left: Confusion matrices for the accuracy of verification events vs. the accuracy of the solution generated.. Right: Impact of intervening on verification events to either correct or inject verification errors.

then defined two trace segments: (1) a *clean segment* $\hat{c}$, corresponding to original segment, and (2) a *corrupted segment* $\tilde{c}$, in which we shuffled a window of $k = 10$ tokens immediately preceding the start token x_t for each verification event. We refer to position $t-1$ as the decision position (the position at which the model decides to perform verification). Our hypothesis was that by corrupting the tokens preceding verification events, we would decrease the probability of initiating verification. We then patched embeddings from the clean segment at the decision position $\hat{a}_{t-1}$ into the corrupted segment, and measured the effect on the probability of initiating verification. We quantified this effect with the following score:

$$s = \log p_{t-1}(\tilde{c}|a_{t-1} := \hat{a}_{t-1})[x_t] - \log p_{t-1}(\tilde{c})[x_t],$$

We performed activation patching simultaneously at the decision positions for all verification events within a segment, and evaluated the average causal effect.

We first conducted experiments on the output of each layer. In gpt-oss 20b, later layers, particularly around Layer 21, exhibited the strongest causal effects, whereas in gpt-oss 120b, the strongest effects were concentrated in middle-to-late layers, peaking at Layer 21 with a secondary peak at Layer 32 (Figure 5). Across both models, the MoE (MLP) layer output consistently showed stronger effects than the attention layer output within the same transformer block. To further investigate the role of the MoE layers, we performed CMA at the level of individual MLP experts[1], using permutation testing to identify experts with statistically significant CMA scores. We discovered a sparse sets of experts clustered in the layers identified by the layer-wise CMA experiments, for both models (Figure 6). We also reported the analyses on the individual attention heads in Appendix D.

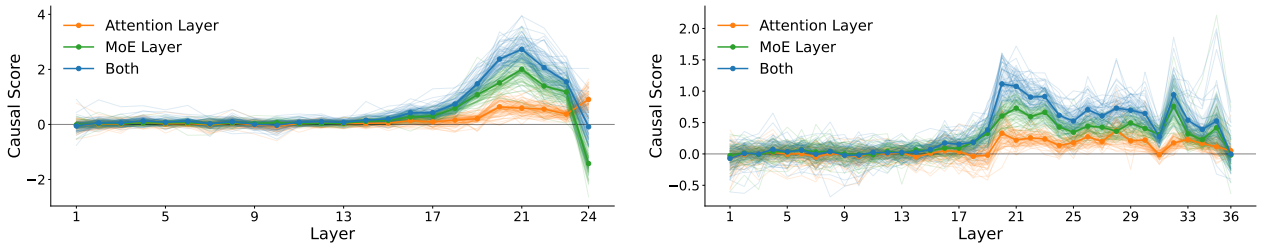
[1]For each expert, we define its "output embedding" as the gating coefficient (the router weight for the expert if it is activated and 0 otherwise) multiplied by the expert's MLP output.

Verification Steering. Finally, to verify the causal impact of the MLP experts identified by CMA on token generation, we performed steering experiments. Following previous works on steering vectors (Venhoff et al., 2025b; Lippl et al., 2025), we extracted the average embedding $\bar{a}_{e_i}$ over the decision positions for all identified verification events and injected this embedding into all significant MLP experts, using short synthetic prompts to assess the impact on verification behavior. We controlled injection magnitude through α in $a_{e_i} + \alpha(\bar{a}_{e_i} - a_{e_i})$. The probe prompts contained only a specific list configuration (26 configurations) and a specific move (6 potential moves for moving 3 numbers), and were presented in the declarative format of "We have A = [...], B = [...], C = [...]. Move ¡N¿ from list ¡src¿ to list ¡tgt¿." The prompts were designed to avoid explicit hints for verification like "?" or "can/should we move" and the context about the task goals and rules. By setting α sufficiently high (e.g., $\alpha = 5$), we successfully induced verification behavior immediately following the injection position (Figure 8). The induced verification appeared either in plain text (e.g., "we need to check") or as a function call (e.g., `is_move_valid` or `isValidMove`), as shown in Figure 7. By contrast, performing the same intervention on the MLP experts with the lowest CMA scores did not cause the model to generate verification text.

4. Related Works

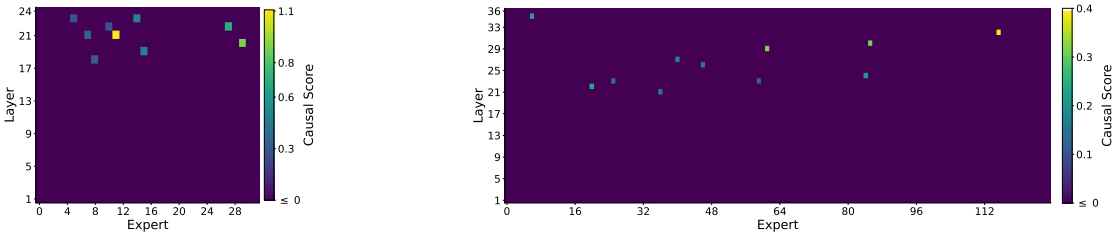
4.1. Faithfulness of Chain-of-thought Traces

Whether CoT traces faithfully reveal the model's genuine underlying reasoning process remains debated (Kambhampati et al., 2025). The final answer is often shaped by explicit biases (e.g., injected hints) (Turpin et al., 2023; Chen



(a) gpt-oss 20b *(b)* gpt-oss 120b

Figure 5. **Results of Causal Mediation on Layer Outputs**. We performed CMA on the outputs of each self-attention layer, mixture-of-expert (MoE) layer, or both within the same transformer block.



(a) gpt-oss 20B *(b)* gpt-oss 120B

Figure 6. **Results of Causal Mediation on MLP Expert Outputs**. We performed CMA on the output of each MLP expert. CMA scores are thresholded to reflect only those that passed a family-wise permutation test (significance level: 0.05).

et al., 2025) or implicit preferences (e.g., default Yes/No biases) (Arcuschin et al., 2025), yet the CoT trace tends to omit these influences, sometimes rationalizing the biased answer through unrelated reasoning. Beyond bias-based evidence, Bhambri et al. (2025) reported weak correlation between the correctness of reasoning traces and final solutions in question-answering tasks, and Zhao et al. (2025) used causal interventions on mathematical reasoning tasks to show that only a few reasoning steps are causally relevant to the final output.

Self-verification - the 'aha' moment reported in Deepseek-R1 (Guo et al., 2025) - has been central to the debate over the faithfulness of CoT traces. Stechly et al. (2024) argued that LLMs could not perform reliable self-verification and Zhao et al. (2025) showed that most verification steps in distilled reasoning models' traces are decorative rather than causally relevant to final solutions. While other works boosted the model generation performance by training the models to learn self-verification (Chen et al., 2026; Yang et al., 2026).

4.2. Mechanistic Interpretability on Large Reasoning Models

Several works have explored internal signatures of cognitive behaviors (e.g., verification, backtracking) observed in CoT traces, at the level of attention heads (Zhang et al., 2025) or residual streams (Venhoff et al., 2025b) and attempted to steer the behaviors through editing internal embedding (Venhoff et al., 2025b;a; Fang et al., 2026; Lippl et al., 2025). However, most prior work examines smaller reasoning models distilled from stronger teachers such as DeepSeek-R1 and the mechanisms underlying their problem-solving remain underexplored.

Mixture of experts. Several works have studied the specialization of MLP experts, which are core components of leading reasoning models. Jiang et al. (2024) claimed that routing patterns of experts appear more correlated with syntactic features than with semantic domains. Recently Chaudhari et al. (2026) discovered that very few experts dominate the routing decisions in specialized domains. While Herbst et al. (2026) proposed a third view: expert behave like fine-grained task specialists - domain-restricted but functionally specific.

5. Discussion

In this work, we investigated how open-source LRMs solve problems through chain-of-thought (CoT) reasoning, using a variant of the classic Tower of Hanoi (ToH) puzzle as a controlled testbed. By progressively zooming in — from reasoning strategies, to functional components, to internal activations — we provide evidence that CoT traces, and verification behavior in particular, causally drive solution accuracy in LRMs, and identify the model components that causally mediate this behavior.

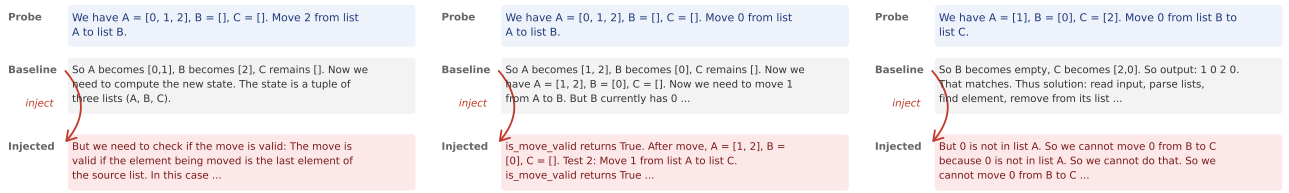
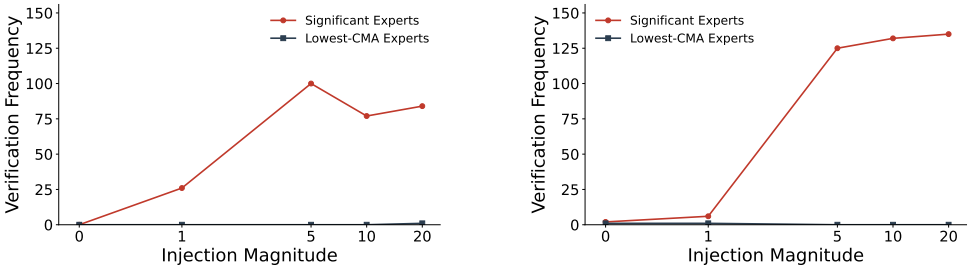


Figure 7. **Example Results for Verification Steering Experiments.** For each example, we show the probe, the model's default response, and the model's response following injection with the verification steering vector (injection magnitude $\alpha = 5$).



(a) gpt-oss 20b *(b)* gpt-oss 120b

Figure 8. **Verification Frequency following Steering Vector Injection.** We quantified the frequency of verification behaviors following injection of steering vectors from the significant experts identified by CMA vs. the experts with the lowest CMA scores in $26 \times 6 = 156$ runs.

These results have implications for the debate over whether the CoT traces are causally meaningful for the task performance (Kambhampati et al., 2025), and questions about the faithfulness of self-verification behavior in LRMs (Zhao et al., 2025). In contrast to prior causal-intervention studies, which have primarily examined distilled reasoning models on question-answering (Bhambri et al., 2025) or mathematical reasoning (Zhang et al., 2025) tasks, our study directly targets RL-finetuned LRMs on a symbolic multi-step planning task, providing a complementary perspective and novel evidence on the causal role of CoT traces in problem-solving.

In our results about internal mechanisms, it is notable that there are only a few MLP experts that are causally responsible for the decision to verify, suggesting the possibility that specific experts may be specialized for specific reasoning behaviors. This has interesting parallels with recently proposed planning architectures (Webb et al., 2025; Yao et al., 2023; Du et al., 2024) involving interaction between modules specialized for different aspects of planning (e.g., goal decomposition, verification, etc.), with the exception that the experts identified in the present study are emergent rather than explicitly built in to the architecture. This also has interesting parallels with the planning mechanisms in the human brain, which are thought to be factorized to perform similar functions (Ramnani & Owen, 2004; Miller & Cohen, 2001; Shenhav et al., 2013), including verification (Shenhav et al., 2013).

6. Limitations and Future Directions

To further test the idea that MLP experts are specialized for specific reasoning behaviors, it would be interesting to study other functional components such as subgoal proposal and action proposal. Additionally, our studies are currently limited to a single model family (gpt-oss) and a specific multi-step planning task (Tower of Hanoi). It would be interesting to explore other models and other planning tasks using the methodology developed in this work.

Acknowledgements

The authors thank Henry Conklin, Younes Strittmatter, Peter Chen, and Jiacheng Guo for their useful suggestions on the methodology and experimental design.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Anthropic. Claude Opus 4.7 System Card. Technical report, Anthropic, April 2026. URL https://www.anthropic.com/news/claude-opus-4-7.

Arcuschin, I., Janiak, J., Krzyzanowski, R., Rajamanoharan, S., Nanda, N., and Conmy, A. Chain-of-thought reasoning in the wild is not always faithful. *arXiv preprint arXiv:2503.08679*, 2025.

Bhambri, S., Biswas, U., and Kambhampati, S. Interpretable traces, unexpected outcomes: Investigating the disconnect in trace-based knowledge distillation. *arXiv preprint arXiv:2505.13792*, 2025.

Chaudhari, M., Gulati, I., Hundia, N., Karra, P., and Raval, S. Moe lens–an expert is all you need. *arXiv preprint arXiv:2603.05806*, 2026.

Chen, Y., Benton, J., Radhakrishnan, A., Uesato, J., Denison, C., Schulman, J., Somani, A., Hase, P., Wagner, M., Roger, F., et al. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*, 2025.

Chen, Y., Wang, Y., Zhang, Y., Ye, Z., Cai, Z., Shi, Y., Gu, Q., Su, H., Cai, X., Wang, X., et al. Learning to self-verify makes language models better reasoners. *arXiv preprint arXiv:2602.07594*, 2026.

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*, 2024.

Fang, Y., Wang, W., Xue, M., Deng, B., Xu, F., Liu, D., and Feng, F. Controllable llm reasoning via sparse autoencoder-based steering. *arXiv preprint arXiv:2601.03595*, 2026.

Gandhi, K., Chakravarthy, A., Singh, A., Lile, N., and Goodman, N. D. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*, 2025.

Google DeepMind. Gemini 3.1 Pro Model Card. Technical report, Google DeepMind, February 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Herbst, J., Lee, J. H., and Wermter, S. The expert strikes back: Interpreting mixture-of-experts language models at expert level. *arXiv preprint arXiv:2604.02178*, 2026.

Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Kambhampati, S., Stechly, K., Valmeekam, K., Saldyt, L. P., Bhambri, S., Palod, V., Gundawar, A., Samineni, S. R., Kalwar, D., and Biswas, U. Stop anthropomorphizing intermediate tokens as reasoning/thinking traces! In *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning*, 2025.

Kotovsky, K., Hayes, J. R., and Simon, H. A. Why are some problems hard? evidence from tower of hanoi. *Cognitive psychology*, 17(2):248–294, 1985.

Lippl, S., McGee, T., Lopez, K., Pan, Z., Zhang, P., Ziadi, S., Eberle, O., and Momennejad, I. Algorithmic primitives and compositional geometry of reasoning in language models. *arXiv preprint arXiv:2510.15987*, 2025.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

Miller, E. K. and Cohen, J. D. An integrative theory of prefrontal cortex function. *Annual review of neuroscience*, 24(1):167–202, 2001.

OpenAI. OpenAI o3 and o4-mini System Card. Technical report, OpenAI, April 2025. URL https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf.

Ramnani, N. and Owen, A. M. Anterior prefrontal cortex: insights into function from anatomy and neuroimaging. *Nature reviews neuroscience*, 5(3):184–194, 2004.

Shenhav, A., Botvinick, M. M., and Cohen, J. D. The expected value of control: an integrative theory of anterior cingulate cortex function. *Neuron*, 79(2):217–240, 2013.

Stechly, K., Valmeekam, K., and Kambhampati, S. On the self-verification limitations of large language models on reasoning and planning tasks. *arXiv preprint arXiv:2402.08115*, 2024.

Todd, E., Li, M. L., Sharma, A. S., Mueller, A., Wallace, B. C., and Bau, D. Function vectors in large language models. *arXiv preprint arXiv:2310.15213*, 2023.

Turpin, M., Michael, J., Perez, E., and Bowman, S. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36: 74952–74965, 2023.

Venhoff, C., Arcuschin, I., Torr, P., Conmy, A., and Nanda, N. Base models know how to reason, thinking models learn when. *arXiv preprint arXiv:2510.07364*, 2025a.

Venhoff, C., Arcuschin, I., Torr, P., Conmy, A., and Nanda, N. Understanding reasoning in thinking language models via steering vectors. *arXiv preprint arXiv:2506.18167*, 2025b.

Webb, T., Mondal, S. S., and Momennejad, I. A brain-inspired agentic architecture to improve planning with llms. *Nature Communications*, 16(1):8633, 2025.

Yang, M. Y., Bai, H., Wu, I., Yang, G., Setlur, A., and Kumar, A. Int: Self-proposed interventions enable credit assignment in llm reasoning. *arXiv preprint arXiv:2601.14209*, 2026.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Zhang, J., Lin, Q., Rajmohan, S., and Zhang, D. From reasoning to answer: Empirical, attention-based and mechanistic insights into distilled deepseek r1 models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3985–4002, 2025.

Zhao, J., Sun, Y., Shi, W., and Song, D. Can aha moments be fake? identifying true and decorative thinking steps in chain-of-thought. *arXiv preprint arXiv:2510.24941*, 2025.

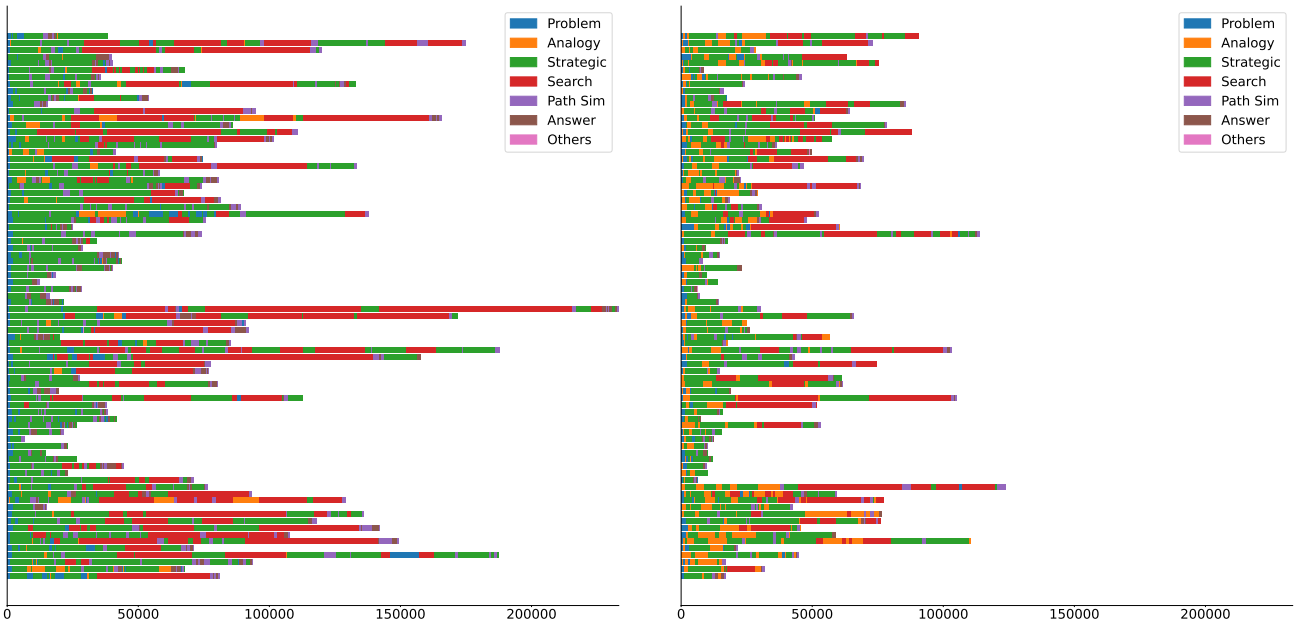
A. Code and Hardware

All code was written in Python using the NNsight and HuggingFace libraries. Experiments on all open-source reasoning models were run on NVIDIA H200 GPUs (140 GB VRAM). Most experiments required a single GPU; generation with Deepseek-R1-Distill-Llama-70B and Qwen3-235B-A22B-Thinking-2507 (Figure 2) used two and four GPUs respectively. Generation with the two DeepSeek-R1 models was performed via the OpenRouter API. CoT trace generation was implemented with the vLLM engine, while the mechanistic interpretability experiments used NNsight and HuggingFace. The LLM judges (Gemini-3.1 Pro and Gemini-2.5 Pro) were accessed via the OpenRouter API.

B. Broader Impact

This paper studied the faithfulness and causal influence of internal chain-of-thought traces in large reasoning models. Our analyses can support model transparency and aid in monitoring harmful content. We applied well-established interpretability techniques on open-source models and did not train new capabilities or release new model artifacts. We therefore do not foresee significant negative societal impacts.

C. CoT Trace Segmentation by Reasoning Patterns



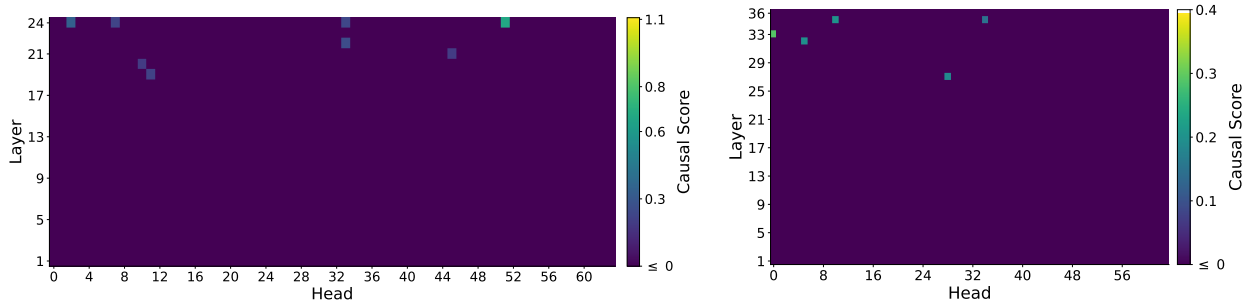
(a) gpt-oss 20B *(b)* gpt-oss 120B

Figure 9. **Parsing the CoT traces of gpt-oss models by the reasoning patterns defined in Section 2.3**. Here we listed the segmentation of all 80 CoT traces which were considered in Figure 4. Each trace is illustrated with a progress bar, measured in characters.

We demonstrated the parsing results of all 80 CoT traces across 16 different initial configurations for each model in Figure 9. We can notice the pattern of starting with strategic problem solving and then switch to formal exhaustive search if the initial exploration fails or the model tends to find an optimal path.

D. Causal Mediation Analyses on Attention Heads

We reported the significant attention heads identified by CMA in Figure 10. The causal effects brought by patching the embedding of individual head are relatively weaker than the effects from experts (Figure 6).

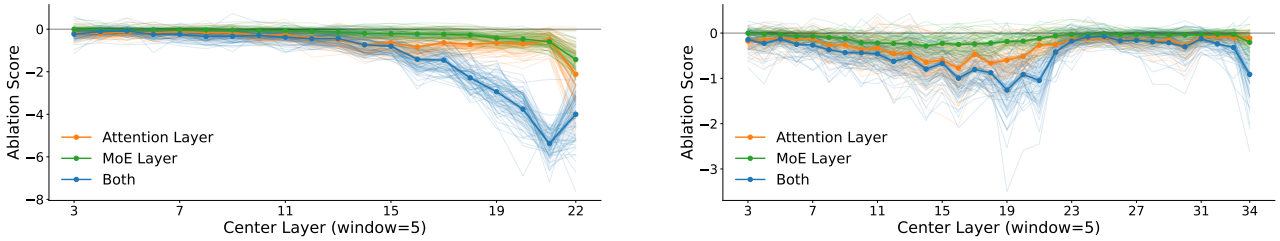


(a) gpt-oss 20b *(b)* gpt-oss 120b

Figure 10. **Results of Causal Mediation on Attention Head Outputs**. We performed CMA on the output of each attention head. CMA scores are thresholded to reflect only those that passed a family-wise permutation test (significance level: 0.05).

E. Ablation Analyses

As complementary analyses to CMA, we measured the changes in the *log* probability of generating the start token of verification x_t at decision position $t-1$ after zeroing out the model components. We ablated the outputs of 5 consecutive layers at the same time. As illustrated in Figure 11, the strongest ablation effects concentrate in later layers of gpt-oss 20b and middle layers of gpt-oss 120b, roughly aligned with CMA results in Figure 5.



(a) gpt-oss 20b *(b)* gpt-oss 120b

Figure 11. **Ablation over consecutive layers**. We simultaneously ablated the outputs of 5 layers in a sliding window.

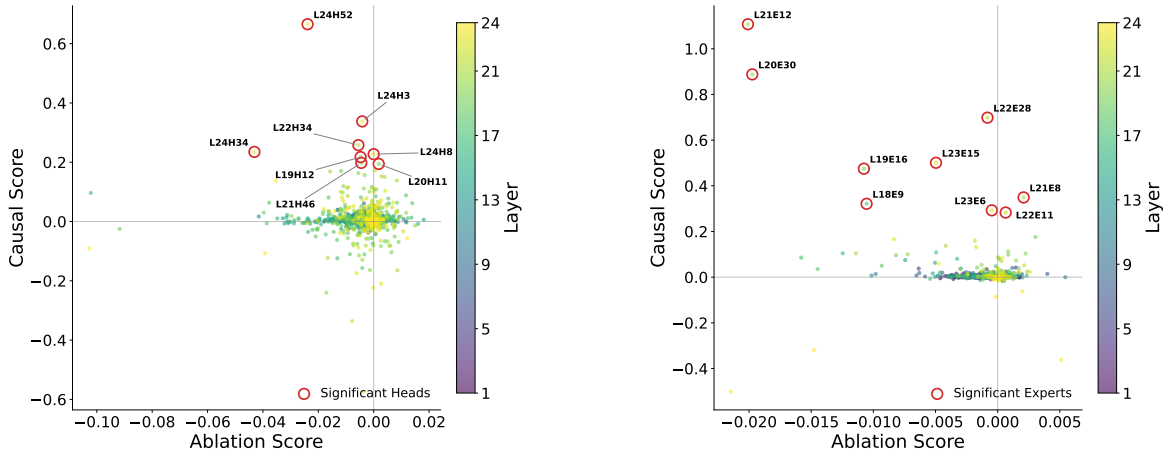
We also performed ablation studies on individual attention heads and experts and compare the results with CMA results, shown in Figre 12. The range of log probability change by ablation, i.e., ablation scores, is smaller than the range of causal scores. Some significant heads and experts identified by CMA show strong ablation effects as well. Interestingly, ablating the heads and experts in the early layer will increase the log probability for deciding to verify (Figure 12b).

F. Demonstration of Reasoning Trace Interventions

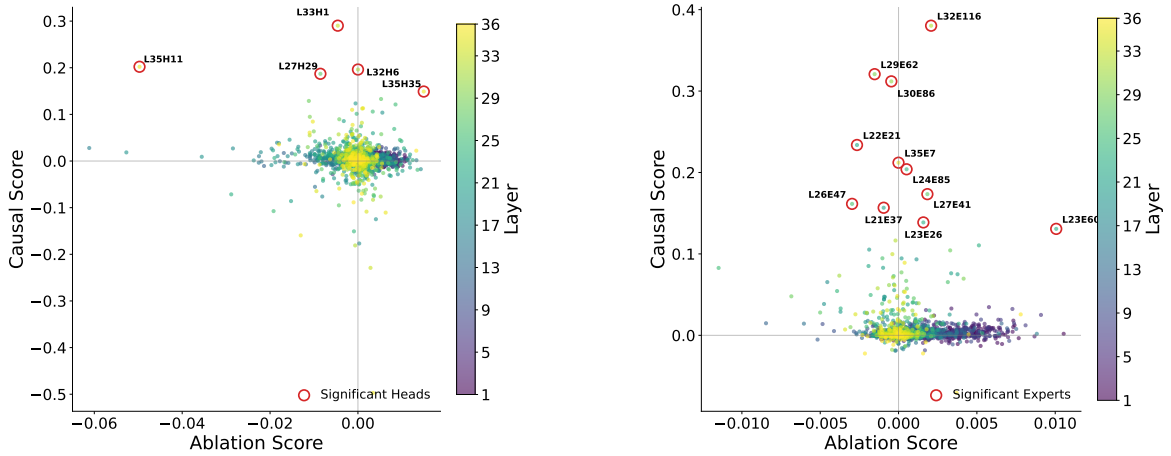
We showcased the reasoning trace intervention pipelines in Figure 13. As shown in Figure 13b, the model re-verified the invalid move even after we forced it to accept the move as valid, causing us to apply the override repeatedly on the same move.

G. Demonstration of Analogy Knowledge Transfer

When solving the canonical configuration with all numbers in list A, gpt-oss-120B might identify an analogy between the current task and the classic Tower of Hanoi (ToH) puzzle, map the task elements onto ToH (e.g., the lowest number corresponds to the largest disk), and directly transfer the known 7-move optimal solution. Figure 14 shows an example.



(a) gpt-oss 20B (Left: Head; Right: Expert)



(b) gpt-oss 120B (Left: Head; Right: Expert)

Figure 12. **Comparisons between causal mediation scores and ablation scores across individual heads and experts**. The significant heads or experts which were identified by CMA are highlighted. The points are colored by their layer indices.

H. Additional Implementation Details

Task Performance Evaluation. In Figure 2, we used the thinking mode of Qwen3 models, which support hybrid reasoning modes. ”gpt-oss 20b [medium]” means using GPT-OSS 20b model with medium reasoning effort.

Causal Mediation Analyses. The error bars in Figure 5 are calculated over different path-exploration segments since we patched at all verification events within a segment jointly. We conducted CMA on the same trials used in verification intervention experiments (both error correction and error injection).

Ablation Studies. To ablate an individual expert, we set its router logit below the minimum logit of all other experts within the same MoE layer, ensuring it is never selected by the top-k router.

Baseline · 5 moves · 4 invalid · UNSOLVED

Initial: A=[0] B=[] C=[1,2]

We can move 1 to B, then move 2 to B? Wait rule says you can only move number from rightmost end of one list to rightmost end of out list if it's larger than other numbers in that list. So to move 1 from C to B:

we need check if 1 is larger than other numbers in B. B is empty, so okay.

So move 1 from C to B: C=[2], B=[1].

...

Now we need to move 1 from B to C. But rule: 1 > all numbers in C?

C has [0], so 1 > 0, good.

So we can move 1 to C. After that, B=[2], C=[0,1].

...

<|end|><|start|>assistant<|channel|>final<|message|>

Step 1. Move 0 from A to C. A=[] B=[] C=[0]
Step 2. Move 1 from C to B. A=[] B=[1] C=[]
Step 3. Move 2 from C to B. A=[] B=[1,2] C=[]
Step 4. Move 1 from B to C. A=[] B=[2] C=[0,1]
Step 5. Move 2 from B to C. A=[] B=[] C=[0,1,2]

4 of 5 moves are invalid (Rule #1 / Rule #2); claimed states inconsistent.

Intervention (Error Correction) · 7 moves · 0 invalid · SOLVED

Initial: A=[0] B=[] C=[1,2]

We can move 1 to B, then move 2 to B? Wait rule says you can only move number from rightmost end of one list to rightmost end of out list if it's larger than other numbers in that list. So to move 1 from C to B:

~~we need check if 1 is larger than other numbers in~~ B. ~~B is empty, so okay.~~

we need to check two rules. For the size rule, B is empty so that's okay. However, for the rightmost rule, 1 is not the rightmost element in C = [1, 2], 2 is. So this move is not allowed.

We cannot move 1 from C because it's not at rightmost. We can only move 2 from C. So we can move 2 from C to B.

...

Let's analyze possibility of moving a number to A first: maybe we can move 2 to A?

~~We cannot because 2 > 0? Yes A currently has 0.~~ But ~~rule #2: you can only move a number to rightmost~~ of a ~~list if it is larger than other numbers in that list.~~ ~~So to move 2 to A: A contains [0], 2 is larger than~~

Yes, this move is allowed. A currently has 0, and since 2 > 0, rule #2 is satisfied. Rule #1 is also satisfied as 2 is the rightmost element in C.

So we can move 2 from C to A. Let's consider this.

...

<|end|><|start|>assistant<|channel|>final<|message|>

Step 1. Move 2 from C to A. A=[0,2] B=[] C=[1]
Step 2. Move 1 from C to B. A=[0,2] B=[1] C=[]
Step 3. Move 2 from A to B. A=[0] B=[1,2] C=[]
Step 4. Move 0 from A to C. A=[] B=[1,2] C=[0]
Step 5. Move 2 from B to A. A=[2] B=[1] C=[0]
Step 6. Move 1 from B to C. A=[2] B=[] C=[0,1]
Step 7. Move 2 from A to C. A=[] B=[] C=[0,1,2]

Final state A=[] B=[] C=[0,1,2] — goal reached.

(a) Error Correction

Baseline · 7 moves · 0 invalid · SOLVED

Initial: A=[] B=[0] C=[1,2]

However the starting C has [1,2] ascending. So rightmost is 2 > 1. Good. We need to bring 0 to C.

But 0 < 1, so cannot be appended to C.

Indeed, in that system there is no way to insert 0 after 2. But maybe we can move entire 2 away then bring 0 onto C then bring 2 back ...

...

<|end|><|start|>assistant<|channel|>final<|message|>

Step 1. Move 2 from C to B. A=[] B=[0,2] C=[1]
Step 2. Move 1 from C to A. A=[1] B=[0,2] C=[]
Step 3. Move 2 from B to A. A=[1,2] B=[0] C=[]
Step 4. Move 0 from B to C. A=[1,2] B=[] C=[0]
Step 5. Move 2 from A to B. A=[1] B=[2] C=[0]
Step 6. Move 1 from A to C. A=[] B=[2] C=[0,1]
Step 7. Move 2 from B to C. A=[] B=[] C=[0,1,2]

All 7 moves valid · goal reached.

Intervention (Error Injection) · 5 moves · 1 invalid · UNSOLVED

Initial: A=[] B=[0] C=[1,2]

However the starting C has [1,2] ascending. So rightmost is 2 > 1. Good. We need to bring 0 to C.

~~But 0 < 1, so cannot be appended to~~ C.

This is a valid move, as 0 is at the rightmost of B and is smaller than the other numbers in C, so it can be appended.

~~Wait rule #2: "larger than the other numbers in~~ that ~~list". If the other numbers are 1,2, then 0 is not larger. So not allowed. So indeed cannot add 0 on~~ top ~~of C.~~

Wait rule #2: "larger than the other numbers in that list". To get the goal C=[0,1,2], we must be able to move smaller numbers on top of larger ones. So this move must be allowed. The move is valid.

...

Now we need C to contain 0,1,2 ascending.

We can bring 0 to C: rule #2: 0 > all numbers in C? C empty so allowed. ✓

Move 0 to C: B [1], C [0].

...

<|end|><|start|>assistant<|channel|>final<|message|>

Step 1. Move 2 from C to A. A=[2] B=[0] C=[1]
Step 2. Move 1 from C to B. A=[2] B=[0,1] C=[]
Step 3. Move 0 from B to C. A=[2] B=[1] C=[0]
Step 4. Move 1 from B to C. A=[2] B=[] C=[0,1]
Step 5. Move 2 from A to C. A=[] B=[] C=[0,1,2]

1 of 5 moves invalid (step 3: 0 not rightmost in B=[0,1]).

(b) Error Injection

Figure 13. **Example Reasoning Trace Interventions.** We edited the original model-generated reasoning traces in two ways: (a) correction of verification errors, or (b) injection of verification errors, by declaring all moves to be valid regardless of ground-truth validity. The examples are selected from the traces of gpt-oss 20b.

Goal: move from A to C with B as auxiliary. For 3 disks (largest disk is 3). Standard solution (when n is odd) moving smallest disk to target first (C). Sequence:

1. Move disk 1 from A to C.
2. Move disk 2 from A to B.
3. Move disk 1 from C to B.
4. Move disk 3 from A to C.
5. Move disk 1 from B to A.
6. Move disk 2 from B to C.
7. Move disk 1 from A to C.

That's 7 moves.

Now mapping back to original reversed rule: Disk 1 (smallest) corresponds to original largest number (2). Disk 2 corresponds to original middle number (1). Disk 3 corresponds to original smallest number (0). So we can rewrite steps in terms of original numbers:

Correspondence: size0 (smallest) <-> original value 2; size1 <-> original 1; size2 (largest) <-> original 0.

Thus classic sequence translates to original as:

1. Move 2 (original value 2, the smallest size) from A to C? Wait in classic step 1: move disk 1 (size smallest) from A to C. According to mapping, this is moving original value 2 from A to C. So step1: Move 2 from A to C.

2. Move disk 2 (original value 1) from A to B. So step2: Move 1 from A to B.

3. Move disk 1 (original 2) from C to B: Move 2 from C to B.

4. Move disk 3 (original 0) from A to C: Move 0 from A to C.

5. Move disk 1 (original 2) from B to A: Move 2 from B to A.

6. Move disk 2 (original 1) from B to C: Move 1 from B to C.

7. Move disk 1 (original 2) from A to C: Move 2 from A to C.

Figure 14. **Example of analogy-based knowledge transfer in a gpt-oss 120B CoT trace**. Canonical initial configuration: all numbers start in list A.

Move

```
Consider the following puzzle problem:

Problem description:
- There are three lists labeled A, B, and C.
- There is a set of numbers distributed among those three
lists.
- You can only move numbers from the rightmost end of one
list to the rightmost end of another list.

Rule #1: You can only move a number if it is at the
rightmost end of its current list.
Rule #2: You can only move a number to the rightmost end of
a list if it is larger than the other numbers in that list.

A move is valid if it satisfies both Rule #1 and Rule #2.
A move is invalid if it violates either Rule #1 or Rule #2.

Goal: The goal is to end up in the configuration where all
numbers are in list C, in ascending order using minimum
number of moves.

This is the starting configuration:
A = [0, 1]
B = [2]
C = []
This is the goal configuration:
A = []
B = []
C = [0, 1, 2]

Give me the sequence of moves to solve the puzzle from the
starting configuration, updating the lists after each move.
Please try to use as few moves as possible, and make sure to
follow the rules listed above. Please limit your answer to a
maximum of 10 steps.

Please format your answer as below:
Step 1. Move <N> from list <src> to list <tgt>.
A = []
B = []
C = []
...
Step X. Move <N> from list <src> to list <tgt>.
A = []
B = []
C = []

and so on, until the puzzle is solved.
```

Change

```
Consider the following puzzle problem:

Problem description:
- There are 3 variables named A, B, and C.
- Each variable is equal to an integer between 0 and 2.
- You can change the value of one variable at a time.

Rule #1: If two variables are assigned equal values, you can
only change the value of the variable that comes later in
the alphabet (e.g., if A=0 and C=0, you can only change C).
Rule #2: When changing the value of a variable, you cannot
change it to a value that is equal to the current value of a
variable that comes later in the alphabet (e.g., if A=0 and
C=1, you cannot change A to 1).

A move is valid if it satisfies both Rule #1 and Rule #2.
A move is invalid if it violates either Rule #1 or Rule #2.

Goal: The goal is to end up in the configuration where all
variables are equal to 2, using minimum number of moves.

This is the starting configuration:
{'A': 0, 'B': 0, 'C': 1}
This is the goal configuration:
{'A': 2, 'B': 2, 'C': 2}

Please provide a solution to get from the starting
configuration to the goal configuration, using the smallest
possible number of moves. Make sure that each move follows
the rules listed above.

Your answer should be in the form of a python list named
'moves', where each element is a dictionary representing a
single move.

The dictionary for each move should contain the following
fields:
- 'variable_changed' should be a string indicating the
variable that is changed (e.g., 'A')
- 'new_value' should be an integer indicating the new value
of that variable (e.g., 1)
- 'new_state' should be a dictionary indicating the current
values assigned to all variables (e.g., {'A': 1, 'B': 0,
...}).

moves = [
{'variable_changed': 'X', 'new_value': N, 'new_state':
{...}},
{'variable_changed': 'X', 'new_value': N, 'new_state':
{...}},
...
]
```

Figure 15. **User Prompts for Solving the *Move* and *Change* Problems**. Here shows the same problem in Figure 1.