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

Anonymous authors
Paper under double-blind review

Abstract

Large reasoning models (LRMs) achieve strong performance that is widely attributed to deliberate thinking in long chains of thought (CoT). Whether the content of these traces is causally relevant to the final answer, however, remains actively debated. In this paper, we present 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 strategies the models use to find solutions, including analogy to a known problem, exhaustive search, and means-ends analysis (MEA)–a strategy that has been identified in the study of human planning. Focusing on MEA episodes, we identified 4 dominant reasoning primitives, including subgoal proposal, action proposal, verification, and backtracking. We intervened on these primitives, strengthening or disrupting them by editing the reasoning trace, finding that these interventions had a strong impact on the accuracy and optimality of the resulting solutions. Using causal mediation analysis, we found that verification could be induced by a steering vector built from the outputs of the Mixture-of-Experts (MoE) layers with the strongest causal effects on the model's output. Together, these results provide causal evidence that the content of CoT traces shape the answers generated by LRMs, and show that these traces can be decomposed into meaningful primitives with reliable mechanistic signatures.

1 Introduction

Chain-of-thought (CoT) reasoning has become a critical component of 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 the high-level reasoning strategies that these models use to solve problems, including diverse strategies such

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.

as analogical reasoning, exhaustive search, and means-ends analysis (MEA). MEA in particular has strong roots in the study of human planning (Newell & Simon, 1972), and forms the basis of the influential General Problem Solver (GPS) (Newell et al., 1959). Second, focusing on MEA, we identified the functional primitives that these models used to carry out this particular reasoning strategy, including subgoal proposal, action proposal, verification, and backtracking, many of which have direct analogs in the GPS formalism. We then tested the causal relevance of these primitives, by intervening at the token level to either enhance or degrade each component. These interventions had a significant impact on both the accuracy and optimality of the solutions generated by the model, demonstrating the causal relevance of these reasoning behaviors. Finally, we investigated the internal mechanisms involved in generating these behaviors, with verification as a case study, identifying a sparse set of causally relevant experts in the Mixture-of-Experts layers from which steering vectors could be extracted and applied to control model reasoning behavior. 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).

In our text-based version of ToH (Figure 2), there are three lists labeled A, B, and C with n numbers distributed across these lists (Figure 2b). The goal is to *move* all numbers to list C, arranged in ascending order, using the fewest moves possible, 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. We selected $n = 3$ for both types of problems, and tested on all $3^3 - 1 = 26$ possible initial configurations.

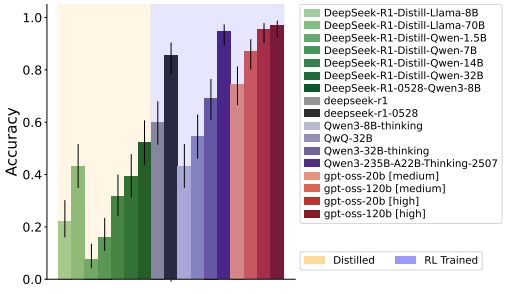
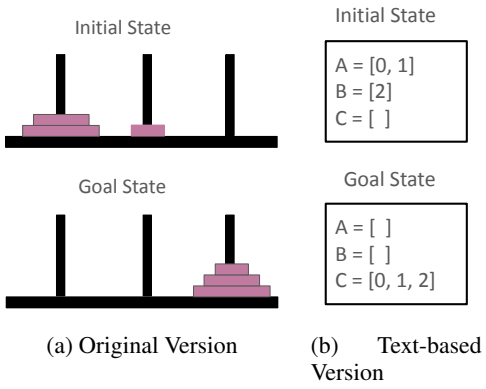


Figure 1: **Large Reasoning Model Performance on Text-based ToH.** Performance across 5 runs each for 26 ToH *Move* problems. Error bars reflect binomial confidence intervals.

We evaluated the task performance of LRMs in terms of the *accuracy* and *optimality* of the final answers. An answer is *correct* if it provides a path from the initial state to the goal state in which each move complies with the rules. An answer is *optimal* if it is correct and uses the fewest moves possible. The optimal paths can be computed for any start configuration by breadth-first search or by some perceptual strategies in Simon (1975); for the initial state in Figure 2, it is 7 moves.

2.2 Model Performance



(a) Original Version (b) Text-based Version

Figure 2: **Tower of Hanoi (ToH) Task Designs.** (a): the original version with physical pegs and disks; (b): the text-based version used in our study, which includes lists and numbers.

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. Figure 1 shows that 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 performing models: all achieving around 95% accuracy, with more than 85% optimal solutions (Figure 9b).

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 the 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.
- **Means-ends Analysis**: In these episodes, the model applied a common human problem-solving strategy called means-ends analysis (Newell & Simon, 1972) to search for a solution in an informal, exploratory manner. These episodes often involve repeatedly decomposing the goal into subgoals by comparing the current state with the goal, as well as other behaviors such as proposing actions, verifying them against rules, and backtracking after hitting the dead end.
- **Exhaustive Search**: In these episodes, the model formalized the task as a state graph and exhaustively 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 was 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.

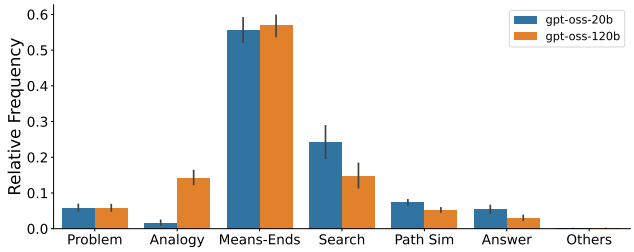


Figure 3: **Distribution of Reasoning Patterns.** Relative frequencies are computed at the character level. We selected the 16 most challenging problems, for which the optimal solution required at least 5 moves, and performed 8 runs per problem. Error bars reflect bootstrap confidence intervals.

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. We also audited the parsing results to assure the reliability of LLM judges (Section H). As illustrated in Figure 3, both models predominantly deployed human-like means-ends analysis. 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 (3/10 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 ineffective. Both models tended to adopt systematic search as a fallback strategy: either to identify a candidate path after means-ends analysis failed, or to find the optimal path using formal methods such as BFS, as shown in Figure 10. The gpt-oss 20b model deployed formal search methods more frequently.

3 Components of Means-ends Analysis in LRMs

Our analysis of LRM reasoning traces indicated that means-ends analysis (MEA) was the dominant strategy used for solving ToH problems. We further identified 4 prominent functional reasoning components from these episodes.

- **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.
- **Backtracking**: Recently taken actions are abandoned, and the task state is reverted to a previously visited configuration (typically when exploration reaches a dead end).

The first three components map closely onto the core operations (also called productions) of the General Problem Solver (GPS), a program that implements MEA, in the simplified form of the production system given by Simon (1975) (Figure 16). Some optimal strategies for solving classic ToH problems, such as the use of a perceptual strategy or a goal-recursion strategy, can be realized as special cases of GPS (Simon, 1975). We also observed that gpt-oss models will sometimes abandon the failed approach, jump back to a previously visited state, and try a different approach after hitting a dead-end. This backtracking behavior is not captured in the original production system formulation but is a common component of the reasoning behaviors that we observed in LRM traces.

To gain a better understanding of the causal role that these components play in supporting the reasoning capabilities of LRMs at both the external token level and internal embedding level, we first intervened on these components through trace editing, either corrupting or repairing them, and quantified the effect on downstream performance (Section 3.1). We then chose to focus on the **verification** component as a case study of researching internal mechanisms that are responsible for these primitives. We extracted steering vectors from the output embeddings of a sparse set of MLP experts, identified by causal mediation analysis, in the Mixture-of-Experts (MoE) layers, and applied them to short probe prompts to induce verification behavior(Section 3.2).

3.1 Reasoning Trace Intervention

We used the same 128 runs of challenging problems as in Figure 3. Within each MEA episode, 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 kept all runs with at least one complete path-exploration segment, and forced the model to generate its final answer after the *first* segment was complete. This lowered the accuracy of gpt-oss-20b and gpt-oss-120b from 94.6% (106/112) and 97.0% (96/99) with the full traces to 55.4% (62/112) and 83.8% (83/99). We intervened on each MEA component within these reasoning segments and measured how either *accuracy* or *optimality* of the resulting answer changed. On average, repairing a primitive raised the optimality rate to $2.3\times$ that of the control and accuracy to $1.6\times$, whereas corrupting reduced optimality to $0.25\times$ and accuracy to $0.43\times$ of the control.

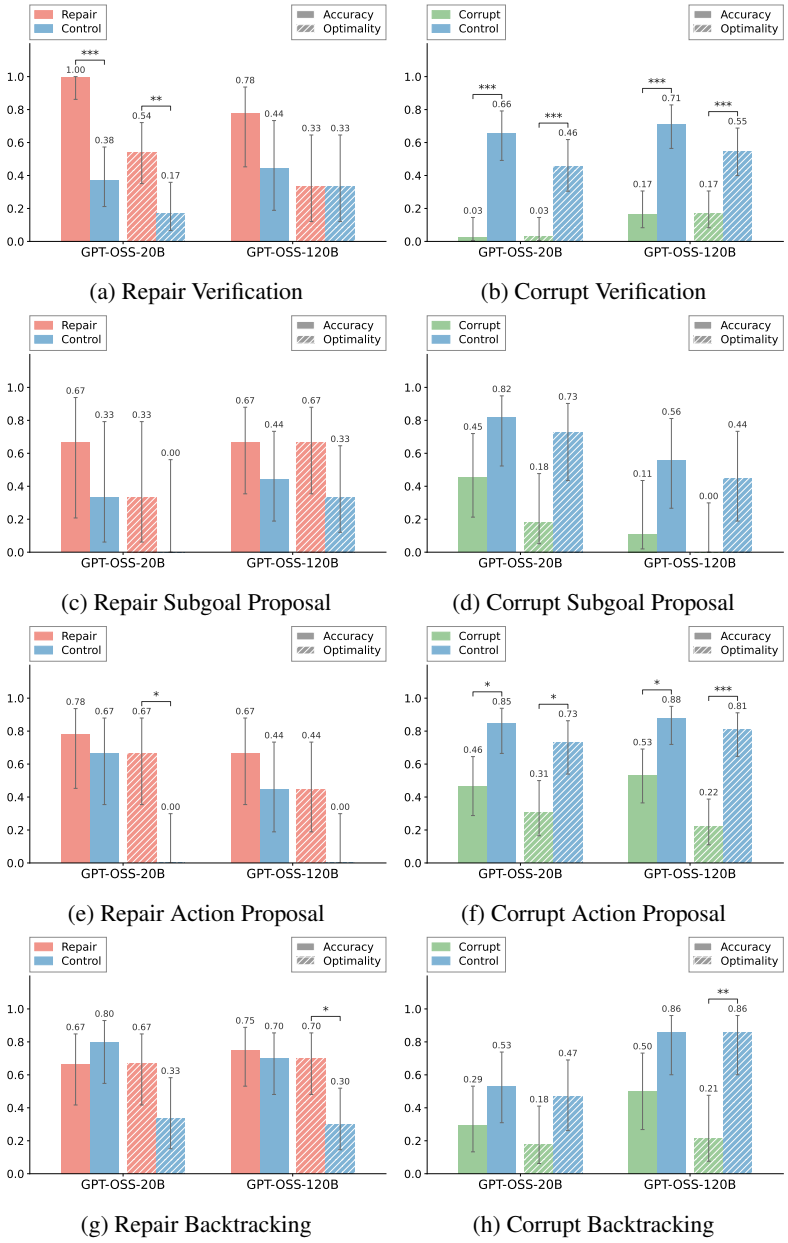
Verification Intervention. We identified all verification errors produced during the first complete path-exploration segments using Gemini-2.5 Pro as a parsing agent. A verification error refers to an incorrect judgment about the validity of the move to be verified, i.e., characterizing an invalid move 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, as illustrated in Table 1. Given this medium-to-strong correlation, we mainly examined the causal impact of verification on answer *accuracy* by manipulating the verification process in three ways:

(1) Repair. 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. The agents' outputs were validated by a human audit (Section H). Figure 13a shows an example of this process.

(2) Corrupt. 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 repair verification) such that all moves were declared to be valid regardless of ground-truth validity.

(3) Control. To rule out effects unrelated to the content of the edit, such as sampling noise, we built a paraphrase control. At every verification that either of the above interventions would edit, we instead replace the original verification with a paraphrase of itself that keeps its original content and approximately matches the length of the intervention's replacement text.

As shown in Figure 4a, repairing verification errors corrected 100% and 78% of the incorrect solutions generated by gpt-oss 20b and 120b respectively compared with 38% and 44% under the control. Conversely, corrupting the verification substantially impaired the accuracy of the final solutions, reducing the accuracy of previously correct solutions to 3% and 17% for gpt-oss 20b and 120b, respectively (Figure 4b), while the control experiments maintained around 70% accuracy (McNemar test, $p < 0.001$).



(a) Repair Verification (b) Corrupt Verification

(c) Repair Subgoal Proposal (d) Corrupt Subgoal Proposal

(e) Repair Action Proposal (f) Corrupt Action Proposal

(g) Repair Backtracking (h) Corrupt Backtracking

Figure 4: **Effects on Answer Accuracy and Optimality by Causal Trace Intervention.** Error bars reflect binomial confidence intervals. * marks significant differences between the intervention (either repair of corrupt) and control (exact McNemar test: $*p < 0.05$, $**p < 0.01$, $***p < 0.001$).

Subgoal Proposal Intervention. Among the subgoals proposed during MEA, we focused on one that is well defined and easy to control: removing an obstacle that blocks an intended move. This subgoal corresponds to production G4.1 of the GPS production system (Figure 16). The key choice in G4.1 is the obstacle `Difference` $= j$, and the optimal perceptual strategy (Figure 17) prescribes the optimal choice: the smaller obstacle (T4′). We therefore selected cases in which more than one obstacle blocks the intended move (e.g., at A = [0, 2], B = [], C = [1], both 2 and 1 block moving 0 to C). Then we intervened on which obstacle j the model chose to remove first (e.g., 2 or 1), and measured how this choice affected the *optimality* of the final path.

We started from the original path-exploration segments which produced correct answers without any verification errors during reasoning and used Gemini-2.5 Pro to locate the moments where the model

picks one of two obstacles blocking an intended move. For **Repair**, we used the segments which produce non-optimal correct paths and edited the moments where the model chose a non-optimal obstacle so that it chose the optimal one instead (following the similar iterative pipeline design as verification intervention). For **Corrupt**, we used the optimal traces and edited the moments where the model chose the optimal obstacle to move, so that it chose another obstacle. For **Control**, we reworded the original content of the moments where the intervention would edit. As shown in Figure 4c, repairing the model's choice of which obstacle to remove turned more non-optimal solutions into optimal ones than the control did. On the other hand, corrupting this choice made more than 80% of the originally optimal traces non-optimal, and many of them no longer reached a correct solution while the control produced a much smaller drop (Figure 4d).

Action Proposal Intervention. Following the same approach as for subgoal proposal, we chose an action-proposal behavior that maps directly onto the GPS production system: grounding an under-specified goal, such as moving an obstacle out of the way, into a specific move. This is production G4.2 (Figure 16), which sets the destination list of an obstacle that blocks an intended move (e.g., at A = [0], B = [1], C = [2], moving 0 to C first requires moving 2 off C). The perceptual strategy (P4′.2 in Figure 17) sends the obstacle to the third list, the one that is neither the source nor the target of the intended move (e.g., B), as the optimal choice. To avoid confounding with verification behavior, we intervened only at moments where the model chose a destination for an obstacle and both other lists were legal destinations, so that either choice gives a valid move. We then measured the effect of the intervention on answer *optimality*. We ran the **Repair**, **Corrupt**, and **Control** experiments with the same pipeline as for subgoal proposal. As shown in Figure 4, repairing the destination turned more than 40% of the non-optimal traces into optimal ones, whereas the control did not recover a single optimal solution. Conversely, corrupting the destination reduced the optimality rate of the originally optimal traces to below 35%, compared with above 70% in the control.

Backtracking Intervention. We defined backtracking as the model jumping back to an earlier state from which it had taken at least two moves to reach its current state. We used the same set of segments as in the subgoal and action proposal interventions. For **Repair**, once the model had made two moves after leaving the optimal path, we inserted a backtrack that returned it to the last state on the optimal path, together with the reason. As shown in Figure 4g, repair turned about 70% of the originally non-optimal solutions into optimal ones, at least twice the rate of the control, while accuracy stayed similar. For **Corrupt**, whenever the model backtracked, we replaced the backtrack with an instruction to keep its previous moves and continue from the current state. This reduced the optimality rate of the originally optimal solutions to about 20%, compared with 47% and 86% in the control in each model, and also lowered accuracy accordingly.

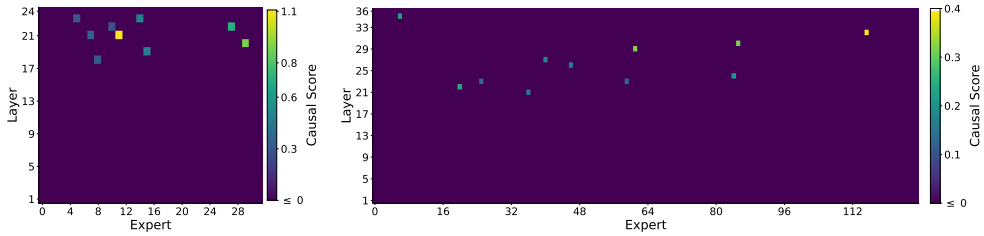
3.2 Mechanistic Interpretability

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 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 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 5). Additional analyses are in Section E.2.

[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.



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

Figure 5: **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).

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 7). 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 6. By contrast, performing the same intervention on the MLP experts with the lowest CMA scores did not cause the model to generate verification text.

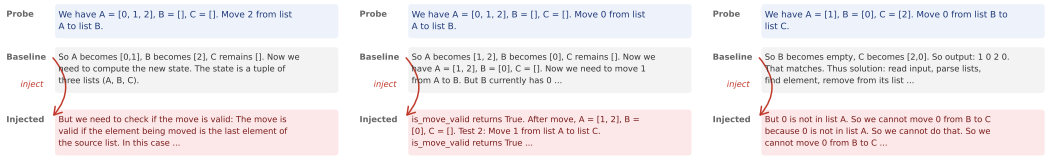
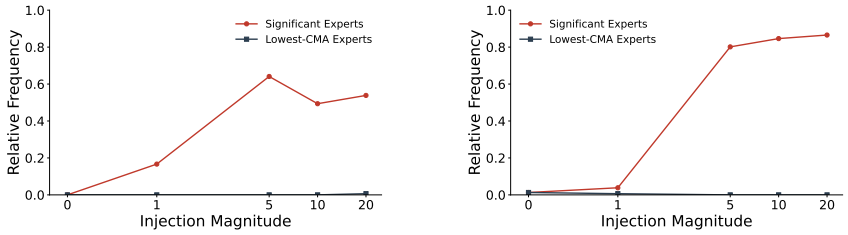


Figure 6: **Example Results for Verification Steering Experiments.** For each example, we show the probe, the model's default response, and the model's response following injection with the verification steering vector (injection magnitude $\alpha = 5$).



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

Figure 7: **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.

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 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.

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 four prominent functional primitives, many of which have direct mapping to GPS formalism of Means-ends Analysis, are presented in CoT traces, and they causally drive solution accuracy and optimality in LRMs, and start to identify the model components that causally mediate the core reasoning primitive verification.

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.

6 Limitations and Future Directions

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. Although we demonstrated strong causal effects of 4 critical reasoning behaviors, which could be mapped to the program implementation of means-ends analysis, on the model performance, it would be interesting to discover the precise algorithm which could incorporate these primitives as components and show the process how the model solves the problem step by step.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, namely as judges to parse or edit reasoning traces, and to help write the code for the experiments under our guidance. We have not used generative AI tools for generating synthetic datasets, helping develop theoretical models or conceptual frameworks, designing or providing feedback on research methodology or experiments, assisting with translation, cleaning or reformatting datasets, supporting qualitative and thematic data analysis, or interpreting results. Formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in writing proofs, and proposing or refining hypotheses are not applicable to this work. Additionally, we used generative AI tools for making minor edits to the draft and identifying relevant literature. We have reviewed all AI-assisted work. We audited all the parsing or editing results given by LLM judges (see Section H), and LLM-generated code was verified and tested for correctness by two authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The research adheres to all guidelines and principles outlined in the ICLR Code of Ethics, including considerations for fairness, accountability, and transparency. Ethical practices were maintained throughout the study, ensuring compliance with all applicable standards.

REFERENCES

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

Iván Arcuschin, Jett Janiak, Robert Krzyzanowski, Senthooran Rajamanoharan, Neel Nanda, and Arthur Conmy. Chain-of-thought reasoning in the wild is not always faithful. *arXiv preprint arXiv:2503.08679*, 2025.

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

Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, et al. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*, 2025.

Yuxin Chen, Yu Wang, Yi Zhang, Ziang Ye, Zhengzhou Cai, Yaorui Shi, Qi Gu, Hui Su, Xunliang Cai, Xiang Wang, et al. Learning to self-verify makes language models better reasoners. *arXiv preprint arXiv:2602.07594*, 2026.

Yi Fang, Wenjie Wang, Mingfeng Xue, Boyi Deng, Fengli Xu, Dayiheng Liu, and Fuli Feng. Controllable llm reasoning via sparse autoencoder-based steering. *arXiv preprint arXiv:2601.03595*, 2026.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. 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/.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Subbarao Kambhampati, Kaya Stechly, Karthik Valmeekam, Lucas Paul Saldyt, Siddhant Bhambri, Vardhan Palod, Atharva Gundawar, Soumya Rani Samineni, Durgesh Kalwar, and Upasana Biswas. Stop anthropomorphizing intermediate tokens as reasoning/thinking traces! In *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning*, 2025.

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

Samuel Lippl, Thomas McGee, Kimberly Lopez, Ziwen Pan, Pierce Zhang, Salma Ziadi, Oliver Eberle, and Ida Momennejad. Algorithmic primitives and compositional geometry of reasoning in language models. *arXiv preprint arXiv:2510.15987*, 2025.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

Allen Newell, J. C. Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pp. 256–264, Paris, 1959. UNESCO.

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.

Herbert A Simon. The functional equivalence of problem solving skills. *Cognitive psychology*, 7 (2):268–288, 1975.

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

Eric Todd, Millicent L Li, Arnab Sen Sharma, Aaron Mueller, Byron C Wallace, and David Bau. Function vectors in large language models. *arXiv preprint arXiv:2310.15213*, 2023.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. 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.

Constantin Venhoff, Iván Arcuschin, Philip Torr, Arthur Conmy, and Neel Nanda. Base models know how to reason, thinking models learn when. *arXiv preprint arXiv:2510.07364*, 2025a.

Constantin Venhoff, Iván Arcuschin, Philip Torr, Arthur Conmy, and Neel Nanda. Understanding reasoning in thinking language models via steering vectors. *arXiv preprint arXiv:2506.18167*, 2025b.

Taylor Webb, Shanka Subhra Mondal, and Ida Momennejad. A brain-inspired agentic architecture to improve planning with llms. *Nature Communications*, 16(1):8633, 2025.

Matthew YR Yang, Hao Bai, Ian Wu, Gene Yang, Amrith Setlur, and Aviral Kumar. Int: Self-proposed interventions enable credit assignment in llm reasoning. *arXiv preprint arXiv:2601.14209*, 2026.

Jue Zhang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 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.

Jiachen Zhao, Yiyou Sun, Weiyan Shi, and Dawn Song. Can aha moments be fake? identifying true and decorative thinking steps in chain-of-thought. *arXiv preprint arXiv:2510.24941*, 2025.

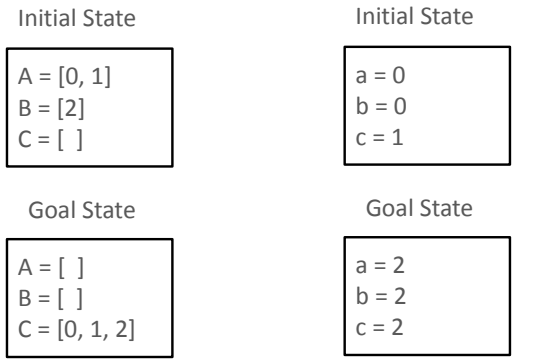
A Code and Hardware

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 1) 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 Performance in Move vs. Change Problems



(a) *Move* Problem (b) *Change* Problem

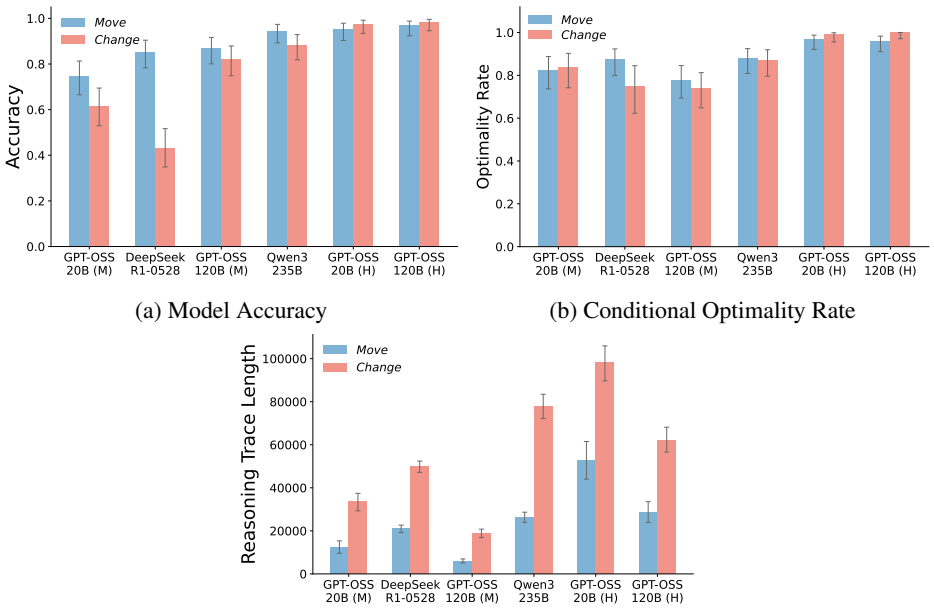
Figure 8: **Two isomorphic text-based versions of ToH.** (a) *Move* problem; (b) *Change* problem.

Inspired by classic work on human problem solving (Kotovsky et al., 1985), we also compared our text-based version with its isomorphic variant, referred to as *Move* problems and *Change* problems.

***Change* problems.** There are n variables named in alphabetical order (a, b, c, ...) each assigned a value of 0, 1, or 2 (Figure 8b). 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 chose the 6 models with the best performance on *move* problems and evaluated their performance on *change* problems (Figure 9) in terms of accuracy, optimality, and CoT trace length. 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 9c). These results suggest that LRMs may employ similar strategies as human problem solvers.



(a) Model Accuracy

(b) Conditional Optimality Rate

(c) Reasoning Trace Length

Figure 9: **Comparison Between Move and Change Problems.** (a): Accuracy on move vs. change problems. Error bars reflect binomial confidence intervals. (b): Conditional optimality rate among all the correct answers. Error bars reflect binomial confidence intervals. (c): Length of reasoning traces for move vs. change problems. Error bars reflect bootstrap confidence intervals. Reasoning trace lengths are counted in characters.

	Answer correct	Answer incorrect
0 Verification errors	48	7
≥ 1 Verification errors	14	43
Total	62	50

(a) gpt-oss 20b

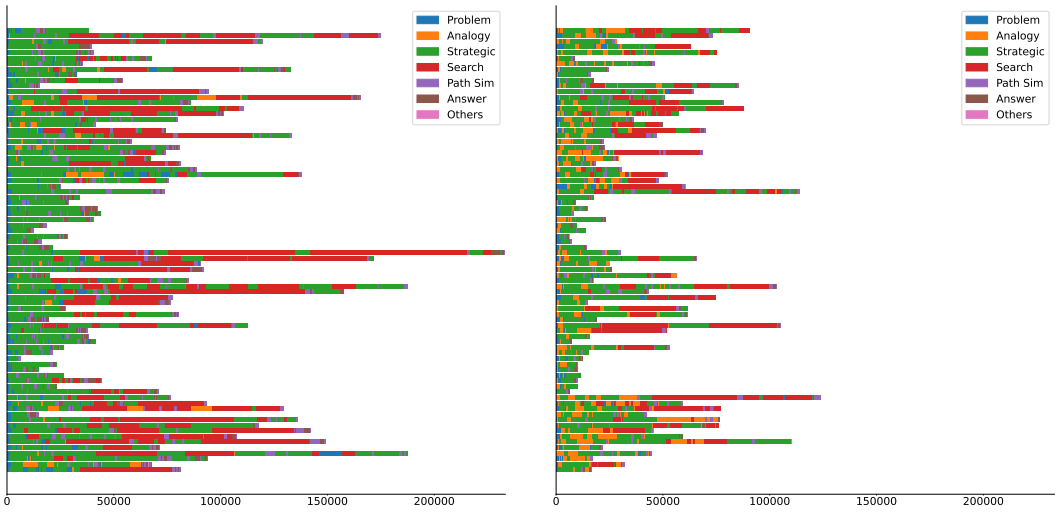
	Answer correct	Answer incorrect
0 Verification errors	70	7
≥ 1 Verification errors	13	9
Total	83	16

(b) gpt-oss 120b

Table 1: **Relationship Between Verification and Accuracy.** Confusion matrices for the accuracy of verification events vs. the accuracy of the generated solution. Correlation: (a) gpt-oss 20b: $\phi = 0.631$, $p = 2.5 \times 10^{-11}$; (b) gpt-oss 120b: $\phi = 0.359$, $p = 3.5 \times 10^{-4}$.

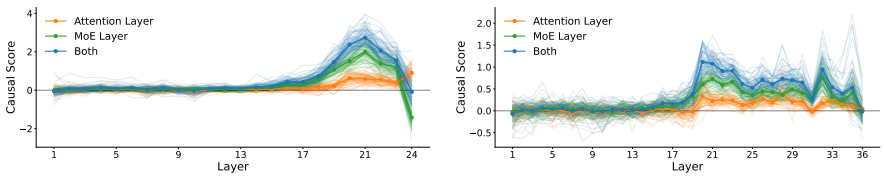
D CoT Trace Segmentation by Reasoning Patterns

We demonstrated the parsing results of all 80 CoT traces across 16 different initial configurations for each model in Figure 10. 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.



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

Figure 10: **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 3. Each trace is illustrated with a progress bar, measured in characters.



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

Figure 11: **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.

E Additional Causal Mediation Analyses

E.1 Layer Level

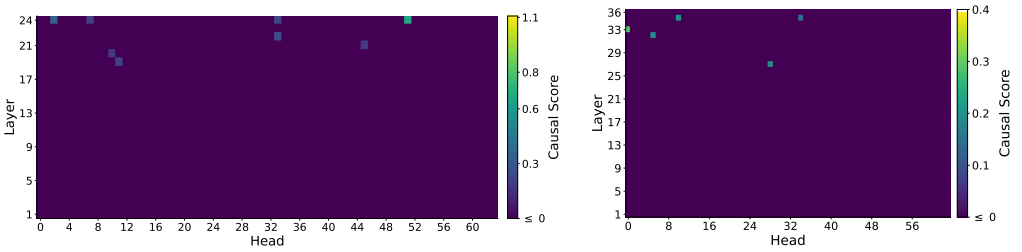
We first conducted experiments on the output of each layer. In gpt-oss 20b, later layers, particularly around Layer 21, exhibited the strongest causal effects, whereas in gpt-oss 120b, the strongest effects were concentrated in middle-to-late layers, peaking at Layer 21 with a secondary peak at Layer 32 (Figure 11). Across both models, the MoE (MLP) layer output consistently showed stronger effects than the attention layer output within the same transformer block.

E.2 Attention Head Level

We reported the significant attention heads identified by CMA in Figure 12. The causal effects brought by patching the embedding of individual head are relatively weaker than the effects from experts (Figure 5).

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.



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

Figure 12: **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).

Table 2: Manual audit of the LLM judge's reasoning-strategy segmentation (Section 2.3).

Model	Audited Episodes	Fully Accepted (label correct, no split)	Sentence-level Acc.	Macro F1	Cohen's κ
gpt-oss-20b	181	169 (93.4%)	0.990	0.977	0.984
gpt-oss-120b	134	127 (94.8%)	0.978	0.952	0.966

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.

H Human Audit on LLM Judge Reliability

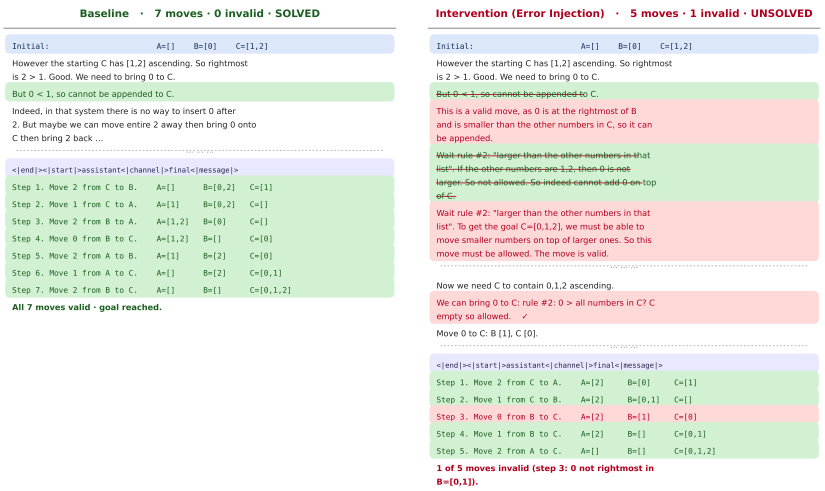
For reasoning-strategy segmentation in Section 2.3. We manually audited the reasoning-strategy episodes produced by the LLM judge (Gemini 3.1). For each model, we selected 10 samples spanning 10 distinct problems (i.e., initial list configurations). For every episode, we independently judged (a) whether the assigned strategy label is correct and (b) whether the content mixes episodes belonging to different strategies; whenever either check failed, we assigned the correct label, or split the episode and labeled the resulting parts. As shown in Table 2, around 94% of episodes were accepted as-is for both models. We then measured the agreement between our corrected annotation and the judge's at the sentence level (each sentence inherits the label of the episode containing it): sentence-level accuracy is approximately 99%, and the macro F1 over all strategy labels lies between 0.94 and 0.97. Inter-rater reliability between our human annotation and the LLM judge's, measured as sentence-level Cohen's κ, is 0.98 (gpt-oss-20b) and 0.97 (gpt-oss-120b).

For the identification of verification events and verification errors in Section 3.1. We manually audited the verification events extracted by the LLM judge. For each model, we randomly sampled 50 detected events across different path-exploration segments (100 in total) and judged, for each identified event: (a) whether it is a genuine verification (b) whether its onset (the decision point) is placed correctly and the span covers the complete check, and (c) whether the extracted move, current state, and declared verdict match the text. 97 of the 100 events are genuine verifications, and the extracted move / state / verdict are correct for all of them (100%). The accuracy of locating decision points is around 84%. There, the imperfect onsets are off by only a phrase or two and always stay within the verification event.

We also manually annotated all verification events in 10 complete episodes per model and computed the recall of the judge's detections: 81.5% (gpt-oss-20b) and 85.4% (gpt-oss-120b) (Table 3). This is expected, as our judge prompts explicitly prioritize precision over recall. We further examined



(a) Error Correction



(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.

Table 3: Manual audit of verification events extracted by the LLM judge (Section 3.1).

Model	Precision	Decision-point (Onset) Acc.	Event Completeness	Field Extraction (Move / State / Verdict)	Recall
gpt-oss-20b	98.0% (49/50)	83.7% (41/49)	85.7% (42/49)	100% (49/49)	81.5% (159/195)
gpt-oss-120b	96.0% (48/50)	85.4% (41/48)	83.3% (40/48)	100% (48/48)	85.4% (181/212)
Pooled	97.0% (97/100)	84.5% (82/97)	84.5% (82/97)	100% (97/97)	83.5% (340/407)

every missed verification: only 1 of 67 contains a verification error, so the missed events would not bias the reported verification error rates.

Furthermore, we audited all 53 error-labeled verification events behind Table 1 (34 for gpt-oss-20b, 19 for gpt-oss-120b; some samples contain several). In our pipeline, an LLM judge extracts each

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 2.

event's move, current state, and declared verdict; a deterministic rule-based checker recomputes the ground-truth validity from the extracted move and state; an error is flagged when the declared verdict disagrees with it. Manual inspection confirms the move and state are extracted correctly in 49/53 events; the other four rest on states only partially observed in the text. And all the extracted declared verdicts faithfully reflect the models' actual conclusions.

For the confirmation of complete path-exploration segments in Section 3.1. We sampled 20 judge-identified path-exploration segments per model (40 in total) and manually checked whether each segment contains the complete exploration ending in a found path from the initial state to the goal state and whether the judge-marked path-found moment is placed correctly: 40/40 pass on both criteria (100%).

I Additional Implementation Details

Task Performance Evaluation. In Figure 1, 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 11 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.

Productions	Conditions		Actions
G1	Difference = Problem-solved	→	Halt
G2	Difference = Applied, Operator = Move(i, P(i), D(i))	→	Delete(Difference), Delete(Operator)
G3	Difference = Null, Operator = Move(i, P(i), D(i))	→	Apply(i), Delete(Difference)
G4	Difference = j, Operator = Move(i, P(i), D(i))	→	Delete(STM), Push[Operator = Move(j, P(j), D(j))]
G4.1	Difference = ***j***, Operator = Move(i, P(i), D(i))	→	Delete(STM), Push[Subgoal = Move(j, P(j))]
G4.2	Subgoal = POP() = Move(j, P(j))	→	Push[Operator = Move(j, P(j), **D(j)**)]
G5	Operator = Move(i, P(i), D(i))	→	Test(Move(i, P(i), D(i)))
G6	Difference = j	→	Push[Operator = Move(j, P(j), Goal-peg)]
G7	else	→	Test(Remaining-difference)
Test(Move(i, P(i), D(i)))			
GT1	Present-situation = Goal	→	Difference ← Problem-solved, Exit
GT2	Applied(Move(i, P(i), D(i)))	→	Difference ← Applied, Exit
GT3	Applicable(Move(i, P(i), D(i)))	→	Difference ← Null, Exit
GT4	else	→	j ← Difference(Present-situation, Operator-input), Difference ← j, Exit
Test(Remaining-difference)			
GT5	(always)	→	j ← Difference(Present-situation, Goal), Difference ← j, Exit

Figure 16: **Mapping Reasoning Components to Productions in GPS Production Systems**. Blue: Subgoal proposal, either by comparing the difference between current state and goal (G6/G7) or removing the obstacle (G4.1); Orange: Action Proposal, grounding the goal of moving out obstacles (G4.2); Purple: Verification (G5). G4 was a production from original Figure 8 of Simon (1975) and here we split it into G4.1 and G4.2. P(i) means the source list of number i while D(i) means its destination list. Productions GT1-GT5 are sub-functions called by G5 and G7.

Productions	Conditions		Actions
P1′	State = Problem-solved	→	Halt
P2′	State = Done, Goal = Move(k, P(k), A)	→	Delete(State), Delete(Goal)
P3′	State = Can, Goal = Move(k, P(k), A)	→	Delete(State), Move(k, P(k), A)
P4′	State = Can't(J), Goal = Move(k, P(k), A)	→	Delete(STM), Goal ← Move(J, P(J), O(P(k), A))
P4′.1	State = Can't(J), Goal = Move(k, P(k), A)	→	Delete(State), Subgoal ← Move(J, P(J))
P4′.2	Subgoal = Move(J, P(J)), Goal = Move(k, P(k), A)	→	Delete(STM), Goal ← Move(J, P(J), **O(P(k), A)**)
P5′	Goal = Move(k, P(k), A)	→	Test(Move(k, P(k), A))
P6′	State = Biggest(J)	→	Goal ← Move(J, P(J), Goal-peg)
P7′	else	→	Test(Biggest-remaining)
Test(Move(X, P(X), A))			
T1	(∀Y) On(Y, Goal-peg)	→	State ← Problem-solved, Exit
T2	On(X, A)	→	State ← Done, Exit
T3	(X = Top(P(X))) & Free(X, A)	→	State ← Can, Exit
T4′	else	→	**J ← Bigger-of[Next-smaller(k, P(k)), Next-smaller(k, A)]**, State ← Can't(J), Exit
Test(Biggest-remaining)			
T5′	(always)	→	**J ← Biggest(Not-on-goal-peg)**, State ← Biggest(J), Exit

Figure 17: **Perceptual Strategy from Simon (1975).** This strategy is an optimal policy for ToH and a realization of the GPS production system in Figure 16. It sets optimal subgoals, such as the largest obstacle to be moved (T4′), and assigns each number an optimal destination, such as sending the obstacle to the peg that is neither the source nor the target of the blocked number (P4′.2). Please be aware that the biggest disc in the original ToH corresponds to the smallest number in our text-based version.