ON REASONING CORRECTNESS DIRECTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We present experimental evidence suggesting that information about whether a model's reasoning is on the right track can be linearly decodable without defining a causally useful direction for steering. To study this, we introduce `ChainSearch`, a controlled arithmetic reasoning task that extends Countdown with explicit search and backtracking. We train a 3-**m**illion-parameter model to solve this task. Its compact size allows us to study internal model correctness directions in detail using linear probes, mean activation differences, and direct optimisation, as part of a rigorous search for internal reasoning-correctness directions. We find that while the model contains statistically significant internal directions of reasoning correctness, none of the resulting directions consistently improves reasoning under causal intervention. This calls into question whether the linear representation hypothesis is sufficient for identifying causally useful directions in the context of reasoning. Our code is made available together with this work.

1 INTRODUCTION

Empirical research on human metacognition suggests that, in structured mathematical problem solving, people can monitor their progress toward a solution, with such judgments predicting subsequent performance (Metcalfe & Wiebe, 1987). The same holds true in human decision making (Yeung & Summerfield, 2012). This raises an interesting question of whether reasoning in language models, much like other properties (Li et al., 2023; Arditi et al., 2024; Genadi et al., 2026), bears hallmarks of the *linear representation hypothesis* (Park et al., 2024).

Recent work also finds that linear probes can predict whether intermediate answers are correct from internal representations (Zhang et al., 2025), but does not investigate causal steering specifically for reasoning correctness. Although prior work has documented limitations in the reliability of linear steering (Tan et al., 2024; Braun et al., 2025), interventions in reasoning models have successfully modulated behaviours such as uncertainty and backtracking (Venhoff et al., 2025), while steering directions encoding behaviours such as reflection and transition have improved answer accuracy (Chen et al., 2025b; Zhuang et al., 2026). These studies show that controlling particular reasoning behaviours can *indirectly* improve answer accuracy. However, *they do not establish whether correctness itself* is encoded in a *causally useful direction* that can reliably guide a model towards the correct solution.

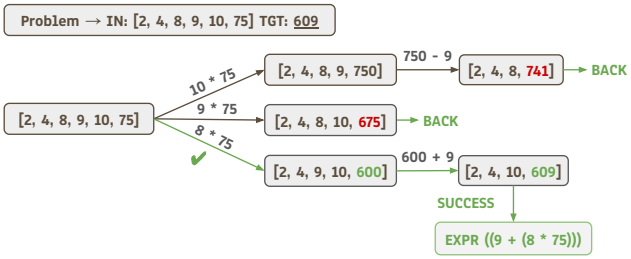


Figure 1: Reasoning often involves trying an approach, recognising when it cannot lead to an answer, and returning to an earlier step to try another. `ChainSearch` makes this process explicit in an arithmetic game, where each step can preserve or remove the possibility of reaching the target. Red numerical results mark states from which the target can no longer be reached without backtracking, while green results mark states where it remains reachable. `BACK` restores an earlier state so another approach can be tried, and `SUCCESS` marks attainment of the target.

One reason why answering this particular question is challenging, is that reasoning correctness must be examined throughout an evolving trace. Compared to properties such as truthfulness and syco-

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.

phancy, which can be evaluated after an entire complete response has been produced (Li et al., 2023; Genadi et al., 2026), reasoning correctness can *change at multiple token positions within a single trace* (e.g. due to backtracking). In other words, we must track whether intermediate states still permit a correct solution through forward operations, or whether the model needs to backtrack to an earlier state from which a solution remains possible. This complicates the choice of where to check for correctness directions, or to steer the model, as there are a wealth of candidate token positions.

This question of when to intervene is accompanied by a localisation problem within the model itself. For reasoning, we must determine not only at which token position a correctness signal is examined along the reasoning trace, but also *where it appears across layers and components*. Zhang et al. (2025) use final-layer correctness signals to decide when to stop generation. However, by the time correctness is detected at this layer, the preceding attention and MLP computations have already taken place, making the final layer a limited intervention site for modifying the internal computations that produced the signal. It is therefore more desirable, when studying reasoning, to be able to investigate directions recovered from deeper parts of the model such as attention heads, residual streams, and MLPs. The resulting number of probe evaluations grows multiplicatively with dataset size, trace length, model depth, and the number of attention heads examined. *Even at the scale of a few billion parameters, dense probing and causal steering can become computationally prohibitive*, particularly when each intervention must be evaluated by generating a new response.

To make such investigation tractable, we draw inspiration from Lee et al. (2024b), Ye et al. (2025b), and Nwadike et al. (2025), who constrain the model and training data to train compact models specialised for particular tasks to isolate particular model capabilities. We construct a similarly controlled setting for reasoning, which we name `ChainSearch`. In this setting, we train compact models *from scratch*, giving us full control over their training, while allowing us to probe correctness signals and test interventions across token positions, layers, and components.

Our main contributions are the following: (1) We introduce `ChainSearch` (see Section 2), a controllable search-based CoT sandbox for studying reasoning correctness in a computationally efficient way. It uses compact transformers trained from scratch on Countdown traces generated by a modified form of the A* algorithm (see Appendix B.2) with explicit state transitions and backtracking. (2) We propose a probing setup that studies reasoning signals across token positions, layers, and model components. This lets us test whether models encode local information about correctness, search direction, state progression, and backtracking. (3) We use `ChainSearch` to test whether reasoning models know when they are on the right track across both successful and failed search trajectories. We find that although correctness is linearly decodable, the recovered directions do not consistently improve reasoning under causal intervention, even when the vectors are directly optimised to increase the likelihood of correct final answers.

2 CHAINSEARCH TASK

`ChainSearch` is a controlled arithmetic search testbed based on the Countdown numbers puzzle (Alliot & Vanaret, 2015; Colton, 2014). The player is given a ***target*** number between `100` and `999`, as well as ***six inputs*** drawn from a pool of 24 numbers (`1` to `10` repeated twice, together with `25`, `50`, `75`, and `100`). The goal is to combine the inputs to reach the target.

Example 1. A `ChainSearch` training trace

```
Problem
IN: [ 3, 6, 10, 10, 50, 75 ] TGT: 425

Solution
[ 3, 6, 10, 10, 50, 75 ] -> 6 * 75 = 450
-> [ 3, 10, 10, 50, 450 ] ; ✗
BACK -> [ 3, 6, 10, 10, 50, 75 ] ;
[ 3, 6, 10, 10, 50, 75 ] -> 10 * 50 = 500
-> [ 3, 6, 10, 75, 500 ] ;
[ 3, 6, 10, 75, 500 ] -> 500 - 75 = 425
-> [ 3, 6, 10, 425 ] ;
SUCCESS ;
EXPR (((10 * 50) - 75)) ✓
```

A legal move is one that selects two values, applies one *operator* from $\{+, -, \times, \div\}$, and uses the result, combined with other numbers remaining, to perform a *state transition*, replacing the two numbers with the result as progress towards the target. For instance, in Example 1, after multiplying 6 and 75 to obtain 450, we transition from `[3,6,10,10,50,75]` to `[3,10,10,50,450]`. Here, an *arithmetic operation* is the calculation applied to two values (as opposed to the operator itself), and a

state transition is the complete update from the preceding state to the resulting state. A "BACK transition" restores an earlier state.

In ChainSearch, there is no requirement to use all the inputs to attain the target, but each input may be used only once, and only positive integers can be produced. The final answer is not the target itself, but the "expression" (EXPR in Example 1) at the end, which should evaluate to the target.

The difference between Countdown and ChainSearch is that ChainSearch introduces an explicit representation of backtracking. Countdown, despite being extensively studied in mathematical and computational settings (see Appendix B.1) was originally developed as a game for humans. To test reasoning in a language model, we introduce BACK tokens to mark when the search abandons a branch and restores an earlier state.

We generate ChainSearch search trajectories using a modified form of A* guided by human-inspired Countdown heuristics (see Appendix I, which details the heuristics used, and a gentler introduction in Appendix B.2). Our autoregressive decoder-only transformer is trained on such data (as shown in Example 1), to produce the entire trace, including the final expression, conditioned only on the "input numbers IN & target TGT". Further examples containing different numbers of BACK transitions are provided in Appendix J. At a high level, the SFT split of ChainSearch contains 1,016,676 examples. We additionally generate 6,948,231 exploratory traces for pretraining. The corresponding data splits are addressed further in Appendix B.3. Additional details on model training and task performance are provided in Appendices B and D.

3 Correctness Direction Analysis

To investigate whether the model contains causally useful correctness directions, we first specify how candidate directions are defined. We then ask whether correctness is linearly decodable and whether the recovered directions remain stable across various dimensions. We construct and analyse these directions using 115,367 generated traces, including a 16,114-trace test split used in full for held-out probe reporting. Section 4 provides a subsequent causal test, evaluating whether interventions along these directions improve reasoning.

3.1 Defining and Studying Correctness Directions

Each candidate correctness direction is defined by four experimental axes. The *target label* (ℓ) specifies the type of correctness, the *semantic/token position* (p) specifies when activations are measured, the *direction method* (m) specifies how the direction is constructed, and the *model component* (c) specifies where inside the model it is measured. We denote the resulting direction by $u_{m,\ell,p,c}$ and describe each axis below

I. Target labels (l):
   a) *Reachability.* This records whether the state produced by an arithmetic state transition, or restored by a `BACK` transition, can still reach the target using only valid forward arithmetic operations. All selected token positions within an arithmetic state transition share the label of its resulting state, while token positions associated with `BACK` share the label of the restored state. This yields an exact intermediate label without requiring explicit intermediate answers.
   b) *Final-answer correctness.* This records whether the complete trace produces a correct final expression. It is attached to each selected token position using the trace's eventual outcome.

   Utilising these two labels helps distinguish local state viability from eventual success.

II. Semantic token positions (p): We consider semantic token positions that represent *key decision points* within the models reasoning trace. Particularly, we look at the first operand position, the second operand position, the operator tokens $+ \; - \; \times \; \div$, the result token for a given operation, and the resulting state of a state transition. We also look at BACK, SUCCESS, and EXPR positions.

III. Direction methods (m):
   a) *Linear-probe vectors.* For each target label, semantic token position, and model component, we train a linear classifier and use its weight vector as the candidate direction.

b) *CAA*. We subtract the negative-class mean activation from the positive-class mean activation.

c) *Direct Optimisation vectors.* We freeze the model M and optimise an additive vector v by minimising the token-level cross-entropy loss $\min_v \mathcal{L}_{\text{ans}}(M(h+v))$ over a valid final-answer sequence (additional details on this control are provided in Appendix F).

We sought to utilise a representative collection of direction vector methods (see Appendix E). We unit-normalise these vectors to allow for a fair comparison of steering strength later on.

IV. Model locations (c): We construct directions at every attention head, residual stream, MLP output, and the final layer-normalised hidden state, following prior work (see Section 1 and Appendix A).

3.2 Linear Probes

Although we also construct directions using CAA and Direct Optimisation in Section 3.3, we begin with linear probing as a relatively lightweight supervised test of whether local reachability and final-answer correctness are linearly decodable from the model's activations. We train probes at each semantic token position and model component, selecting all hyperparameters on validation data and reporting performance exclusively on the held-out test split.

Figure 2 shows how final-answer decodability develops over the full reasoning trace. Held-out AUROC is already above chance early in generation and, after a small initial dip, steadily increases as the trace progresses. This pattern is similar across all model locations c, although attention heads lag slightly at the starting and ending.

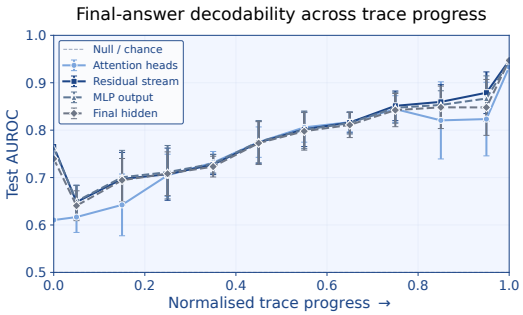


Figure 2: By normalising reasoning trace lengths, we compare how well linear probes predict final-answer correctness at different stages of generation. Correctness is already decodable well above chance early in the trace and generally becomes more decodable as generation proceeds. This pattern appears across attention heads, residual streams, MLP outputs, and the final hidden representation, with AUROC reaching approximately 0.94 by the end. Error bars show one standard deviation around the mean.

Because traces containing many `BACK` transitions contribute more intermediate positions, we repeat the analysis after restricting the data to traces containing at most eight `BACK` transitions. Appendix C Figure 9 closely reproduces the trajectory of increasing linear probe AUROC, indicating that the increase towards the end of the trace is not primarily driven by unusually long or heavily backtracking traces.

These results establish that the model's activations contain substantial information about both local reachability and eventual success that can be decoded with a linear probe. However, even if a linear classifier recovers a correctness direction involved in the model's subsequent reasoning, that direction may not best support effective causal steering. We therefore examine whether alternative construction methods recover different candidate directions. Specifically, we construct additional directions using CAA and Direct Optimisation.

3.3 Beyond Linear Probes (CAA & Direct Optimisation)

Having so far examined correctness through linear probes, we now broaden the search for candidate correctness directions using CAA and Direct Optimisation. This allows us to have several methods for subsequent steering analysis in Section 4.

Introducing CAA and Direct Optimisation creates a comparison challenge. In particular, probe-based vectors arise from a linear classifier, whose held-out AUROC provides a direct measure of discriminative performance. CAA and Direct Optimisation however, do not, by construction, yield an equivalent classification score. We therefore define five direction stability metrics that allow the resulting directions to be compared without treating them as classifiers. Table 1 summarises these metrics over the $\ell \times p \times m \times c$ dimensions introduced in Section 3.1.

Table 1: The five comparisons metrics A–E above, are used to test whether reasoning correctness forms a stable linear direction. Here, $u_{m,p,c}$ denotes the unit-normalised direction obtained using method m at semantic position p and model component c, $\ell \in \{\text{reach}, \text{answer}\}$ denotes the label family, and K is the number of selected semantic positions.

Metric	Measure	Question
A. Successive-position stability	$S^{\mathrm{S}}_{m,\ell,p\to p',c} = \cos\left(u^{\ell}_{m,p,c}, u^{\ell}_{m,p',c}\right)$	What if we go one step at a time across successive reasoning positions?
B. Cross-position stability	$S^{\mathrm{P}}_{m,c} = \dfrac{K\left\|\frac{1}{K}\sum_{p=1}^{K} u_{m,p,c}\right\|^2 - 1}{K-1}$	Does final-answer correctness share a common direction across reasoning positions?
C. Cross-label stability	$S^{\mathrm{L}}_{m,p,c} = \cos\left(u^{\text{reach}}_{m,p,c}, u^{\text{answer}}_{m,p,c}\right)$	Do local reachability and eventual final-answer correctness share the same direction?
D. Cross-method stability (Type 1)	$S^{\mathrm{M1}}_{m_1,m_2,p,c} = \cos(u_{m_1,p,c}, u_{m_2,p,c})$	Does the correctness direction remain stable across construction methods at the same position and component?
E. Cross-method stability (Type 2)	$S^{\mathrm{M2}}_{p,c} = \left\|\mathrm{Proj}_{\mathrm{span}\{u^{P}_{p,c}, u^{C}_{p,c}\}} u^{D}_{p,c}\right\|^2$	Does the Direct Optimisation vector lie within the plane spanned by Probe and CAA?

A) *Successive-position stability:* Successive-position stability S^{S} measures whether a direction persists between neighbouring semantic token positions in the reasoning trace. Using this metric, Figure 3 shows that reachability directions change least from `BACK` to the restored state at attention heads, with CAA generally more stable than Probe. However, since this may simply reflect high certainty that `BACK` transitions tend to result in the same state (a reachable one), additional observations are required. Indeed, Appendix Figure 12 shows a similar attention-head advantage during arithmetic state-transitions. Figure 4 applies the same analysis to final-answer directions, where Probe and CAA-based vectors again exhibit their greatest stability at attention heads.

B) *Cross-position stability:* Cross-position stability S^{P} asks whether final-answer correctness shares a common direction across the wider reasoning trace for a fixed construction method and model component. We report this measure only for final-answer correctness, since Direct Optimisation is not used together with reachability (see Section F). Appendix Figure 5 shows that CAA is most stable under this measure, with attention heads again exhibiting the greatest stability for the Probe and CAA methods.

C) *Cross-label stability:* Cross-label stability S^{L} measures whether reachability and final-answer correctness share a direction at the same token position. Appendix Figure 13 shows that the attention-head "advantage" again extends from positional stability to agreement across correctness labels, particularly for CAA-based vectors. Even there, however, the cosine similarities remain moderate, indicating partial rather than complete agreement between the labels (a phenomenon addressed in further details in the concluding "Observation" to this subsection).

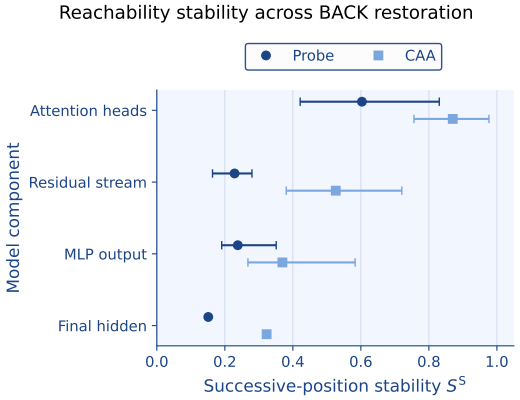


Figure 3: Reachability directions remain more stable across `BACK` and state restoration at attention heads than at other model components. CAA has higher median stability than Probe in every component class. Points show median successive-position stability (S^{S}), and horizontal bars show interquartile ranges.

D–E) *Cross-method stability:* Pairwise cross-method stability S^{M1} tests whether directions constructed using different methods agree at the same token position and model component. Because pairwise cosine cannot determine whether the Direct Optimisation direction lies within the span of the Probe and CAA directions, the second type of cross-method stability S^{M2} measures this as well. Figure 11 combines these two metrics into a single plot. We may observe that Residual-stream and MLP directions remain close to the random projection baseline, whereas attention heads exhibit a broader distribution. However, high S^{M2} scores are seen in a sparse subset of attention-head–position pairs (also interpreted in the concluding "Observation" below).

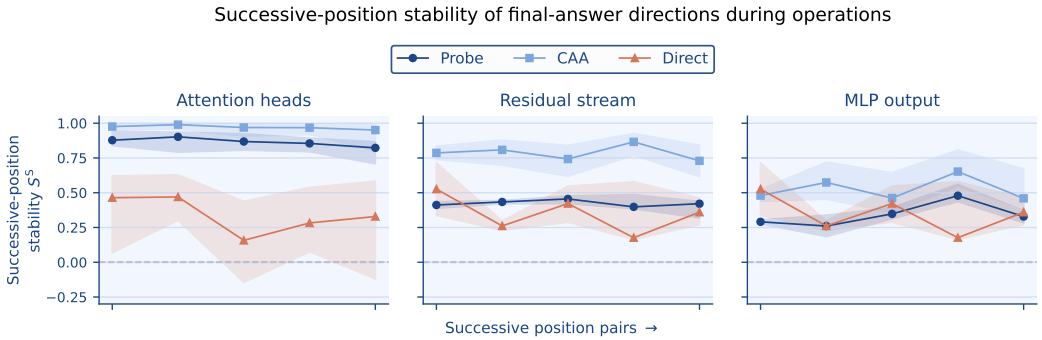


Figure 4: Probe and CAA final-answer directions remain highly stable across successive token positions within arithmetic state transitions at attention heads, whereas Direct Optimisation yields weaker and more variable alignment. Lines show median successive-position stability (S^{S}) within each component class, and shading shows interquartile ranges. An expanded version is provided in Appendix Figure 10.

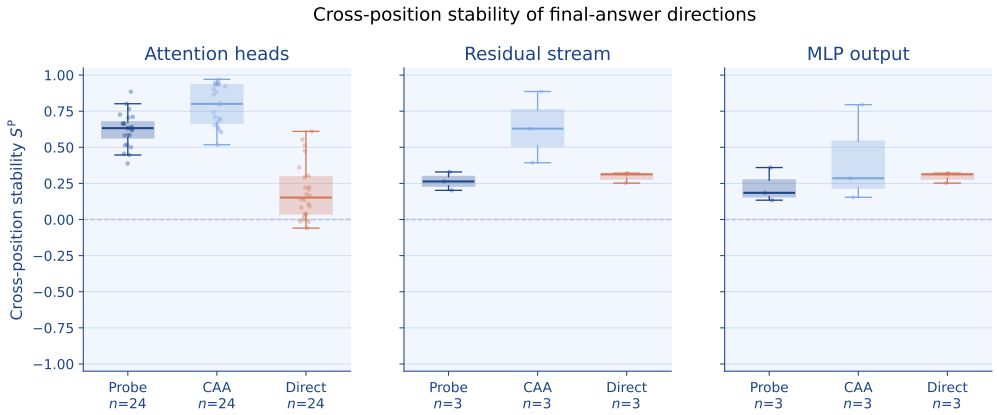


Figure 5: We measure cross-position stability (S^{P}) to test whether final-answer directions remain consistent across the selected positions throughout the reasoning trace. CAA final-answer directions have the highest median cross-position stability in every component class. Probe directions are also more stable across positions at attention heads than at other components, whereas Direct Optimisation directions show no comparable advantage. Each point reports stability at one model location.

Observation: Across the five measures, attention heads most consistently exhibit the clearest directional stability. Indeed, the *successive-position and cross-position stability* results could, on their own, suggest that final-answer correctness follows a common direction that persists throughout reasoning, particularly for Probe and CAA-based directions at those attention heads. *Incomplete alignment across labels and methods does not inherently negate this token position stability observed within each method.* Reachability and final-answer correctness are related but distinct measures of correctness, so the moderate alignment observed for CAA at attention heads, with cosine similarities around 0.5 (corresponding to angles of approximately 60°), is plausible. Nor should distinct construction methods necessarily recover identical vectors in a high-dimensional activation space. The broader spread at attention heads in Figure 11 therefore remains notable. Primarily, these results further motivate the need to compare multiple correctness-direction methods during steering,

rather than relying on a single method as some prior works may have opted to. Section 4 proceeds accordingly.

4 CAUSAL STEERING

While Section 3 examines whether correctness is linearly decodable and geometrically stable, this section asks whether the resulting directions can causally improve reasoning.

We evaluate steering across token position p, correctness label ℓ, construction method m, model component c, and a new dimension intervention strength β, yielding the search space $p \times \ell \times m \times c \times \beta$. The number of steered attention heads, and the intervention frequency, are also studied separately. Candidate intervention configurations are selected using validation data and evaluated on a held-out test split, with each steered generation paired with an unsteered generation of the same problem.

The two correctness label values ℓ naturally allow us to examine causal effects both immediately after an intervention, and at the end of the generated trace:

1. *Next-state reachability* records whether the first state generated after an intervention can still reach the target using only valid forward arithmetic operations. For example, consider the `ChainSearch` problem "`IN: [3,6,10,10,50,75] TGT: 425`". The legal move `6*75=450` produces `[3,10,10,50,450]`, which cannot reach the target without backtracking. By contrast, `10*50=500` produces `[3,6,10,75,500]`, from which `500-75=425` remains possible. We count a problem as improved if the next state is unreachable without steering but reachable with steering, and as worsened if the reverse occurs.
2. *Final-answer correctness* records whether steering changes the correctness of the completed answer. For the same problem, "`IN: [3,6,10,10,50,75] TGT: 425`", an unsteered response ending in `EXPR ((6*75)-10)` evaluates to 440 and is incorrect. If steering instead produces `EXPR ((10*50)-75)`, which evaluates to 425, we count the problem as improved. If an originally correct answer becomes incorrect under steering, we count it as worsened. Unlike next-state reachability, this measures whether the model actually reaches a correct answer, rather than whether a solution remains possible.

Unlike probe reporting, causal steering requires fresh *autoregressive generation* for every intervention condition. We therefore use a staged evaluation procedure in which 256 or 512 paired held-out problems provide an initial indication of whether an effect is statistically sufficient to justify evaluation across the full 16k-sample test split.

We begin with attention heads, which Section 3 identifies as the most favourable test case. Section 4.1 presents steering results for different construction methods across several attention-head counts, while Appendix Section H.1 tests whether these conclusions hold under different head-selection strategies, direction signs, and token positions, as well as whether steering with local-reachability directions affects final answers. Section 4.2 then broadens the analysis to model components beyond attention heads and compares different intervention frequencies. Appendix Section H.2 complements this discussion with additional experiments on intervention strength and vector normalisation, and documents how steering affects trace properties.

4.1 ATTENTION-HEAD INTERVENTIONS

We first test whether correctness directions at the most promising attention heads improve reasoning on held-out problems. Figure 6 compares Probe, CAA, and Direct Optimisation across the highest-ranked head sets, and interventions over all available heads. We vary both the number of steered heads and whether intervention strength is fixed in total or applied separately to each head. Appendix Figure 14 compares steering at attention heads selected using CAA separation with steering at randomly selected heads across several head counts. Neither selection strategy makes steering consistently improve next-state reachability or final-answer correctness.

Across the configurations in Figure 6, steering changes individual outcomes in both directions without producing a consistent benefit. The largest displayed number of problems with improved final-answer correctness is 5 out of 256. CAA improves these 5 outcomes while worsening 4 others,

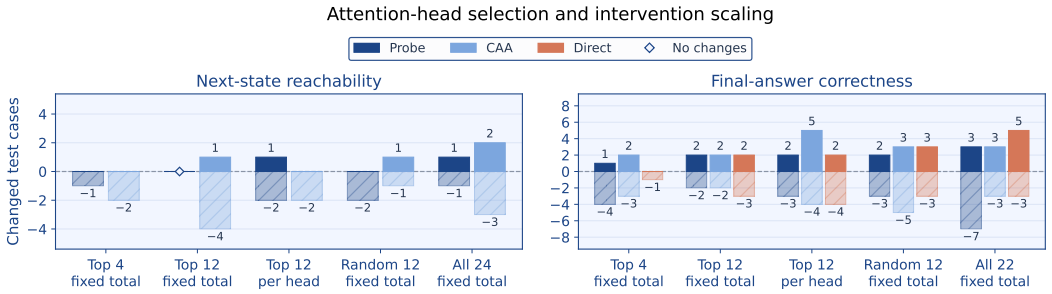


Figure 6: Changing the number of steered attention heads yields no consistent improvement in next-state reachability or final-answer correctness. Each condition compares the same 256 test problems with and without steering. Solid upward bars count problems improved by steering, hatched downward bars count problems worsened, and diamonds indicate no change.

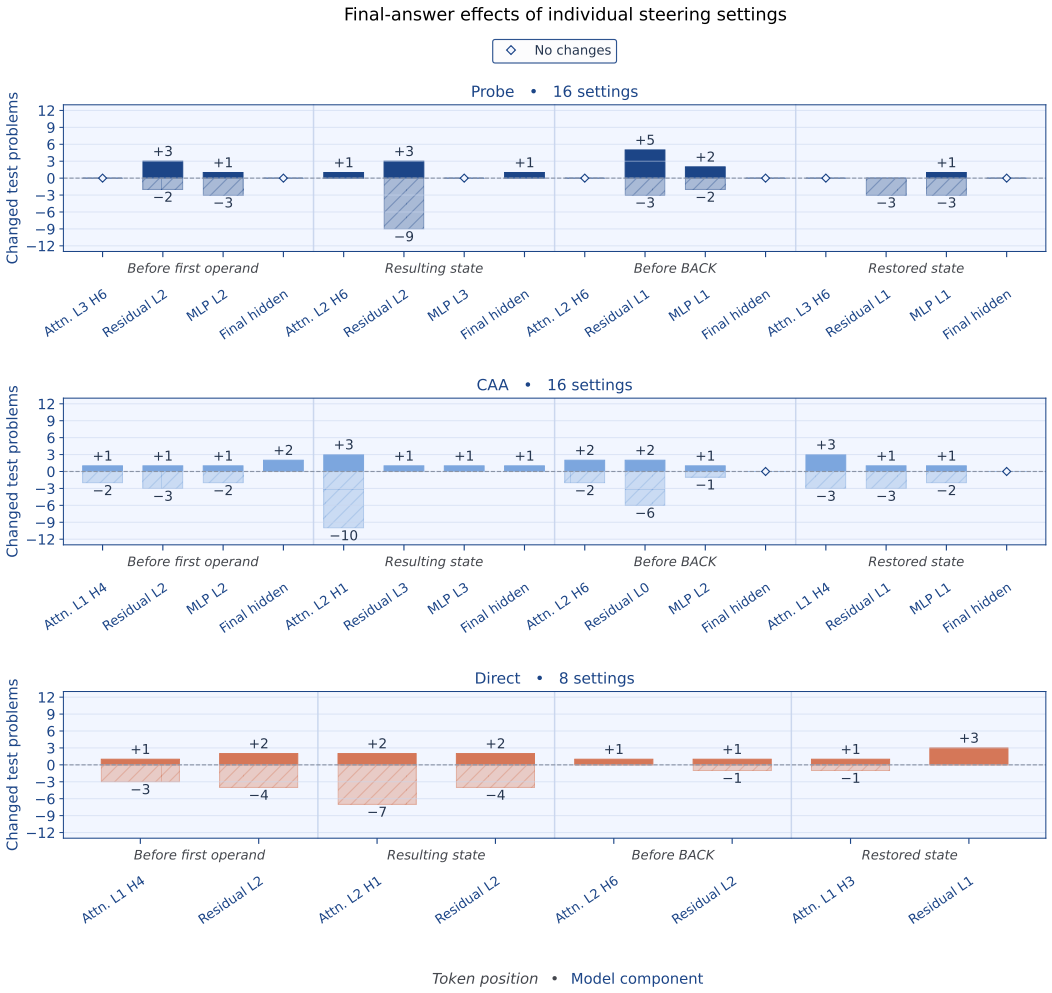


Figure 7: Steering produces small, mixed changes in final-answer correctness, with several of the largest effects worsening outcomes. Upward bars count problems where the model originally gave an incorrect answer but gives a correct one after we apply steering and let it generate a new response. Downward bars count the opposite, where an originally correct answer becomes incorrect. Even Direct Optimisation, which explicitly trains a vector to increase the likelihood of a correct final answer (see Appendix F), does not consistently improve correctness when the model generates a new response. Settings are chosen using validation data, then evaluated on 256 test problems, each with and without steering.

and Direct Optimisation improves 5 while worsening 3. Probe steering across all 22 attention heads instead improves 3 outcomes and worsens 7. Changes in next-state reachability are similarly infrequent and occur in both directions. Appendix Figures 15 and 16 likewise show that neither direction type nor direction sign consistently improves final-answer correctness across intervention positions.

In other words, varying which heads are steered, how many are steered, or the direction sign yields no dependable improvement in either next-state reachability or final-answer correctness.

4.2 Broader Steering Interventions

Building on the attention-head results in Section 4.1, Figure 7 compares Probe, CAA, and Direct Optimisation vectors across token positions and model components. Across these dimensions, steering moves some outcomes toward correctness and others away from it, without consistently improving held-out final-answer correctness.

Figure 8 shows that repeated and multipoint steering change more generations but mainly increase the number of problems with worsened outcomes. Appendix Figures 18 and 19 extend this analysis, showing that neither varying intervention strength nor retaining the raw magnitude of Direct Optimisation vectors produces more reliable control.

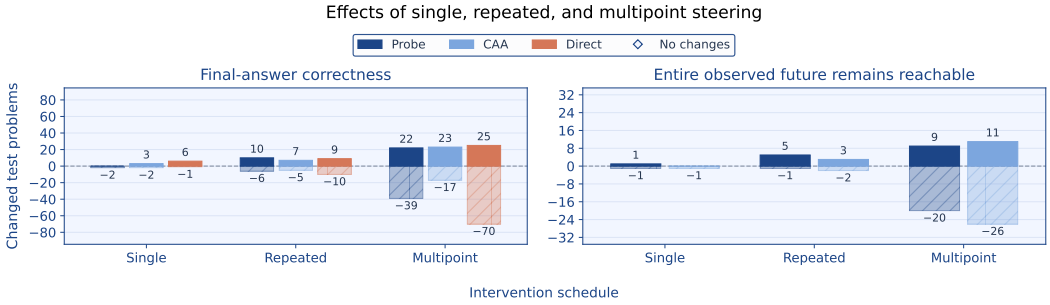


Figure 8: Applying steering more frequently changes more outcomes, but does not consistently improve them. The largest increases in the number of worsened outcomes occur when steering is applied at multiple positions. Single steering acts before the first arithmetic state transition, repeated steering acts before every arithmetic state transition, and multipoint steering additionally acts after resulting and restored states, with settings chosen separately for each position. The panels measure final-answer correctness and whether every subsequent state remains reachable. Positive bars count problems improved by steering, while negative bars count problems worsened. Each setting is evaluated on 512 test problems, with and without steering.

Importantly, these opposing effects do not mean that the interventions are causally inert. Appendix Figure 17 shows that steering changes generation length, `BACK` usage, and expression parseability without producing a common behavioural signature of improved reasoning.

5 Conclusion

We conduct a systematic study of reasoning correctness directions, asking whether linear decodability implies causal usefulness for steering. To do so, we introduce `ChainSearch`, a controlled arithmetic task in which reasoning traces can backtrack while searching for a solution. This simple setting allows us to analyse correctness directions across multiple dimensions and evaluate them through steering interventions. Correctness is linearly decodable, but our analysis reveals a nuanced picture of what constitutes a correctness direction. A direction that appears to capture correctness along one dimension may not do so along another. Even along dimensions where the recovered directions are largely stable, steering with them yields no consistent improvement. This suggests that familiar approaches to linear steering may not fully capture the representational structure relevant to improving reasoning accuracy, motivating new techniques focused on nonlinear representational geometries specific to *reasoning correctness*.

AI USE STATEMENT

This research was developed by the authors through extensive discussion. We used AI primarily to assist with coding/debugging and to help polish the writing of the manuscript. The authors manually reviewed all work with great care and caution.

REPRODUCIBILITY STATEMENT

We provide the code for dataset generation, model training, probing, and causal steering via the zip file in supplementary materials.

REFERENCES

Jean-Marc Alliot and Charlie Vanaret. (the final) countdown. In *GCAI 2015. Global Conference on Artificial Intelligence*, volume 36 of *EPiC Series in Computer Science*, pp. 14–26, 2015. URL https://easychair.org/publications/paper/2L76.

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. In *Advances in Neural Information Processing Systems*, volume 37, pp. 136037–136083, 2024. doi: 10.52202/079017-4322. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/f545448535dfde4f9786555403ab7c49-Abstract-Conference.html.

Joschka Braun, Carsten Eickhoff, David Krueger, Seyed Ali Bahrainian, and Dmitrii Krasheninnikov. Understanding (Un)reliability of steering vectors in language models. In *ICLR 2025 Workshop on Foundation Models in the Wild*, 2025. URL https://openreview.net/forum?id=qGCp2AYosf.

Yuanpu Cao, Tianrong Zhang, Bochuan Cao, Ziyi Yin, Lu Lin, Fenglong Ma, and Jinghui Chen. Personalized steering of large language models: Versatile steering vectors through bi-directional preference optimization. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1567. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/58cbe393b4254da8966780a40d023c0b-Abstract-Conference.html.

Runjin Chen, Andy Arditi, Henry Sleight, Owain Evans, and Jack Lindsey. Persona vectors: Monitoring and controlling character traits in language models, 2025a. URL https://arxiv.org/abs/2507.21509.

Runjin Chen, Zhenyu Zhang, Junyuan Hong, Souvik Kundu, and Zhangyang Wang. SEAL: Steerable reasoning calibration of large language models for free. In *Proceedings of the Conference on Language Modeling (COLM)*, 2025b. URL https://openreview.net/forum?id=klPszYDIRT.

Simon Colton. Countdown numbers game: Solved, analysed, extended. In *Proceedings of the AISB Symposium on AI and Games*, 2014. URL https://doc.gold.ac.uk/aisb50/AISB50-S02/AISB50-S2-Colton-paper.pdf.

Countdown Wiki. Numbers game — techniques. https://wiki.apterous.org/Standard_Method, 2024. Accessed 2025-09-11.

DataGenetics. Countdown game show (numbers round), 2014. URL https://datagenetics.com/blog/august32014/. Accessed 2025-09-11.

Daniel Defays. *Numbo: A Study in Cognition and Recognition*. Basic Books, 1995. URL https://www.researchgate.net/publication/262363566_Numbo_a_study_in_cognition_and_recognition.

Jacob Dunefsky and Arman Cohan. One-shot optimized steering vectors mediate safety-relevant behaviors in LLMs. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=teW4nIZ1gy.

Rifo Ahmad Genadi, Munachiso Samuel Nwadike, Nurdaulet Mukhituly, Tatsuya Hiraoka, Hilal AlQuabeh, and Kentaro Inui. Sycophancy hides linearly in the attention heads. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6896–6912, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.324. URL https://aclanthology.org/2026.eacl-long.324/.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136. URL https://ieeexplore.ieee.org/document/4082128.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.

Michael Katz, Harsha Kokel, and Sarath Sreedharan. Seemingly simple planning problems are computationally challenging: The countdown game. *arXiv preprint arXiv:2508.02900*, 2025. URL https://arxiv.org/abs/2508.02900.

Andrew Lee, Xiaoyan Bai, Itamar Pres, Martin Wattenberg, Jonathan K. Kummerfeld, and Rada Mihalcea. A mechanistic understanding of alignment algorithms: A case study on DPO and toxicity. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 26361–26378. PMLR, 2024a. URL https://proceedings.mlr.press/v235/lee24a.html.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. In *International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=dsUB4bst9S.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 41451–41530, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/81b8390039b7302c909cb769f8b6cd93-Abstract-Conference.html.

Janet Metcalfe and David Wiebe. Intuition in insight and noninsight problem solving. *Memory & Cognition*, 15(3):238–246, 1987. doi: 10.3758/BF03197722. URL https://doi.org/10.3758/BF03197722.

Munachiso S. Nwadike, Zangir Iklassov, Toluwani Aremu, Tatsuya Hiraoka, Benjamin Heinzerling, Velibor Bojkovic, Hilal AlQuabeh, Martin Takáč, and Kentaro Inui. Library-like behavior in language models is enhanced by self-referencing causal cycles. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25365–25377. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1232. URL https://aclanthology.org/2025.acl-long.1232/.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 39643–39666. PMLR, 2024. URL https://proceedings.mlr.press/v235/park24c.html.

Countdown Resources. Countdown resources — 6 small, 2024. URL https://countdownresources.wordpress.com/6-small/. Accessed 2025-09-11.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering Llama 2 via contrastive activation addition. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15504–15522, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.828. URL https://aclanthology.org/2024.acl-long.828/.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/f3c5e56274140e0420baa3916c529210-Abstract-Conference.html.

Daniel Tan, David Chanin, Aengus Lynch, Brooks Paige, Dimitrios Kanoulas, Adrià Garriga-Alonso, and Robert Kirk. Analysing the generalisation and reliability of steering vectors. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/fb3ad59a84799bfb8d700e56d19c231b-Abstract-Conference.html.

Constantin Venhoff, Iván Arcuschin, Philip Torr, Arthur Conmy, and Neel Nanda. Understanding reasoning in thinking language models via steering vectors. In *Workshop on Reasoning and Planning for Large Language Models*, 2025. URL https://openreview.net/forum?id=OwhVWNOBcz.

Wikipedia contributors. Countdown (game show) — numbers round. https://en.wikipedia.org/wiki/Countdown_(game_show)#Numbers_round, 2025. Accessed: 2025-09-11.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract.html.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *International Conference on Learning Representations*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c1b3d1e2cf53bb28cabd801bd58b3521-Abstract-Conference.html.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=Tn5B6Udq3E.

Nick Yeung and Christopher Summerfield. Metacognition in human decision-making: Confidence and error monitoring. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1594):1310–1321, 2012. doi: 10.1098/rstb.2011.0416. URL https://doi.org/10.1098/rstb.2011.0416.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=O6I0Av7683.

Haomin Zhuang, Hojun Yoo, Xiaonan Luo, Kehan Guo, and Xiangliang Zhang. Reliable control-point selection for steering reasoning in large language models. *arXiv preprint arXiv:2604.02113*, 2026. URL https://arxiv.org/abs/2604.02113.

APPENDIX

A EXTENDED RELATED WORK: SELF-CORRECTNESS SIGNALS

A key distinction from prior work is our focus on whether reasoning correctness itself is associated with internal linear direction vectors. Venhoff et al. (2025) instead steer higher-level reasoning behaviours such as uncertainty, example testing, and backtracking. Chen et al. (2025b) and Zhuang et al. (2026) show that steering higher-level reasoning behaviours such as reflection and transition can indirectly improve answer accuracy, but the underlying directions encode those behaviours rather than *correctness itself*.

One study in this direction is by Zhang et al. (2025), who train linear probes to predict this intermediate answer correctness in a reasoning trace. A notable drawback of their setup is that it considers reasoning trace activations at the *last model layer*. This last-layer focus is also seen in the work of Ye et al. (2025b), who study GSM8K-style arithmetic traces with linear probes. However, the last model layer comes *after* the model has completed all core computations relevant to its top-k token ranking, which leaves much of the model under-examined with regards to causal steering effects.

Indeed, Zhang et al. (2025) avoid *causal steering* altogether, and instead use their linear probes for early-exit, to decide when to halt token output. However, this experimental design decision leaves an evidence gap regarding their central question as to whether reasoning models know when they are right.

Another concern is that Zhang et al. (2025) exclude unfinished generations from parts of their analysis. Their early-exit setup allows at most 10,000 tokens per problem. In our AIME 2025 reproduction with DeepSeek-R1-Distill-Qwen-7B, 56.7% of reasoning traces exceeded this limit. This leaves the behavior of these unfinished cases unresolved.

Test-set contamination is another concern. Because MATH was public before the evaluated models were trained, an early-exit probe may partly detect memorized answer patterns rather than online correctness. One way to rule out this concern completely would be to *train the model from scratch* with full control over its training data. However, training a language model on a reasoning task from scratch is non-trivial, and is therefore an important component of the gap in the research literature that `ChainSearch` addresses.

Causal steering, in any case, may take on a number of definitions, which we discuss in greater detail in Appendix Section E. In brief, this work uses probe-based, mean-difference, and directly optimized activation steering. For *probe-based steering*, the steering vector is learned by minimizing a binary prediction loss, $v^* = \arg\min_v \sum_i \mathcal{L}(y_i, v^\top h_i + b)$, and is then used directly as the intervention direction (Li et al., 2023; Genadi et al., 2026). We compare probe-based steering with *mean-difference steering*, whereby we adapt contrastive activation addition (CAA) (Li et al., 2023; Rimsky et al., 2024) using a direction constructed by contrasting correct and incorrect traces $v = \mathbb{E}_i[h_i \mid \text{correct}] - \mathbb{E}_i[h_i \mid \text{incorrect}]$. For completeness, we also consider *directly optimized activation steering*, where the steering vector v is obtained as $v^* = \arg\min_v \sum_i \mathcal{L}(M(h_i + v), y_i)$ (Cao et al., 2024; Dunefsky & Cohan, 2025). Such direct optimization is far more *computationally expensive* (and less common) than the other two methods, requiring repeated forward and backward passes through the model. `ChainSearch` makes it practical because our model has only 3 million parameters rather than billions or trillions.

To keep our steering findings relevant beyond the `ChainSearch` setting, we also restrict interventions to localized model components such as attention heads, residual streams, and MLPs (Li et al., 2023; Chen et al., 2025a; Lee et al., 2024a; Genadi et al., 2026). Model-wide interventions, while feasible in our 3-million parameter setting, may be less extensible to larger models and datasets.

B Dataset Details

B.1 ChainSearch Complexity

As mentioned in Sections 1 and 2, `ChainSearch` is based on the six-input version of Countdown, in which numbers must be combined to reach a target. The four-input version of the game, known as "Game of 24", has become a common testbed for language-model reasoning (Yao et al., 2023; Stechly et al., 2025). The six-input version used in `ChainSearch` has also recently been used as a reasoning benchmark (Ye et al., 2025a). Unlike Game of 24, Countdown permits larger input sets and variable targets. Its state space grows rapidly with the number of inputs, and its general decision problem is NP-complete (Katz et al., 2025). For six inputs, the state-space bound derived by Katz et al. (2025) is over $5{,}400$ times larger than for the four-input Game of 24 setting.

More generally, the search space grows exponentially with the number of inputs. The six-input version of Countdown used in `ChainSearch` contains approximately 1.2×10^7 solvable instances (Colton, 2014). For a fixed input such as `[3,6,10,10,50,75]`, there are $6! \cdot \mathrm{Catalan}(5) \cdot 4^5 = 30{,}965{,}760$ ways to order the inputs, group them with parentheses, and choose an operator for each calculation before removing equivalent constructions or invalid calculations. For example, `((75 * 6) - (50 / 10)) - (10 + 3)` and `((75 - 50) * (10 + 6)) + (10 * 3)` are two different fully parenthesised expressions constructed from the same six inputs. Thus, even a single compact problem permits millions of possible arithmetic constructions.

`ChainSearch` retains this six-input setting while representing search explicitly through state transitions and `BACK`. We use the HR3 system to enumerate all solvable instances (Colton, 2014), then generate supervision using search traces by solving each instance with a modified variant of A* (Hart et al., 1968), as explained in Appendix B.2 below.

B.2 Constructing Search Traces with Modified A*

To construct the search traces used to fine-tune `ChainSearch`, we solve Countdown instances with a modified form of A* (Hart et al., 1968). The HR3 solver described in Appendix B.1 provides final expressions, whereas A* lets us construct structured traces showing arithmetic progress toward the target, unsuccessful attempts, and revisions via `BACK`.

Note: A* is used only to construct the supervised finetuning (SFT) data. We *do not* assume that the trained model internally implements A*, or that it explicitly stores alternative search states. The model learns to do reasoning *entirely from a simple cross-entropy loss on the resulting SFT reasoning traces of how solve the input problems.*

A* is used to generate SFT reasoning traces using a cost function:

$$f(n) = g(n) + h(n),$$

where $g(n)$ measures the cost already incurred and $h(n)$ estimates the remaining cost of reaching the target. In our setting, $g(n)$ counts the arithmetic operations already taken, while $h(n)$ scores the current arithmetic state using the human-inspired strategies listed in Appendix I. Combined, these two terms encourage principled search traces in which the next branch is selected using both the path already taken and an intuitive estimate of the remaining distance to the target, rather than through brute-force exploration.

We make two *modifications* when applying A* as a tool for generating `ChainSearch` data:

First, conventional Countdown solutions record only the chosen arithmetic operations and do not explicitly represent abandoned attempts. We therefore *linearise* the usual branching search of A* into a sequence of state transitions, inserting `BACK` whenever the procedure abandons an unproductive branch and returns to an earlier state.

Second, we do not require $h(n)$ to be *admissible*. In standard A*, admissibility means that $h(n)$ never overestimates the true remaining cost, which supports a guarantee that the resulting path is optimal. Our objective is not to recover the shortest possible trace, but to generate informative search behaviour shaped by strategies used by human Countdown players. We therefore retain the $g(n) + h(n)$ scoring framework while utilising the Countdown-specific heuristics of Appendix I.

B.3 DATASET SPLITS

Table 2: Allocation of `ChainSearch` data across model pretraining, supervised fine-tuning, interpretability analysis, and causal steering. Split counts describe distinct traces or examples, whereas steering counts describe paired condition–problem evaluations. The same held-out problem may therefore be evaluated under several independently selected steering conditions.

Stage	Size	Purpose
Model pretraining		
Exploratory pretraining corpus	6,948,231 traces	Procedurally generated search with controlled backtracking
Pretraining train	6,947,727 traces	Model pretraining
Pretraining validation	504 traces	Pretraining checkpoint selection
Supervised fine-tuning and held-out allocation		
Complete `ChainSearch` corpus	1,016,676 examples	Supervised fine-tuning and held-out analyses
Fine-tuning train	808,033 examples	Model training
Fine-tuning validation	900 examples	Checkpoint selection
Fine-tuning test	9,000 examples	Final model evaluation
Held-out pool for correctness-direction analysis	115,367 examples	Set aside before fine-tuning for correctness-direction analysis
Unassigned held-out reserve	83,376 examples	Reserved for target-disjoint splitting (see Appendix Section D)
Correctness direction analysis		
Generated traces for correctness-direction analysis	115,367 traces	Shared source for Probe, CAA, Direct Optimisation, and evaluation
Train split	83,354 traces	Correctness direction construction
Validation split	15,899 traces	Candidate and hyperparameter selection
Test split	16,114 traces	Held-out probe reporting and steering source pool
Labelled token position instances	6,877,165 instances	Position-level correctness supervision
Cached activation positions	6,335,310 positions	Probe fitting and CAA mean-difference construction
Causal steering evaluation		
Steering test per condition	256 or 512 problems	Initial paired causal evaluation
Complete held-out steering suite	68,096 paired evaluations	236 selected conditions across four experiments

Table 2 summarises how data is allocated across model training, correctness direction evaluation, and causal steering.

Probe results are reported on the full 16,114-trace test split. Steering evaluations instead use 256 or 512 problems per selected condition, because each condition requires fresh autoregressive generation. These subsets provide an initial test of whether an intervention produces effects large and consistent enough to warrant evaluation on the full test split.

Example 2 illustrates why target-disjoint splitting is necessary, as problems sharing a target can produce nearly identical reasoning traces even when their inputs differ slightly. Further details on the splitting procedure are provided in Appendix Section D.

Because targets are kept disjoint across splits, examples must be allocated in target-level groups. Constructing the fine-tuning, evaluation, and interpretability subsets at the designated sizes therefore leaves some examples unassigned. We retain these examples as the "Unassigned held-out reserve" reported in Table 2.

Appendix J provides more example of `ChainSearch` traces containing one to eight backtracks.

Example 2. Nearby inputs can share the same three-backtrack solution structure

```
A ChainSearch Problem
IN: [ 1, 2, 6, 8, 25, 50 ] TGT: 202

[ 1, 2, 6, 8, 25, 50 ] -> 8 * 25 = 200 -> [ 1, 2, 6, 50, 200 ] ;
BACK -> [ 1, 2, 6, 8, 25, 50 ] ;
[ 1, 2, 6, 8, 25, 50 ] -> 6 * 25 = 150 -> [ 1, 2, 8, 50, 150 ] ;
BACK -> [ 1, 2, 6, 8, 25, 50 ] ;
[ 1, 2, 6, 8, 25, 50 ] -> 6 * 50 = 300 -> [ 1, 2, 8, 25, 300 ] ;
BACK -> [ 1, 2, 6, 8, 25, 50 ] ;
[ 1, 2, 6, 8, 25, 50 ] -> 8 * 25 = 200 -> [ 1, 2, 6, 50, 200 ] ;
[ 1, 2, 6, 50, 200 ] -> 2 + 200 = 202 -> [ 1, 6, 50, 202 ] ;
SUCCESS ;
EXPR ((2 + (8 * 25))) ✔

A Different Problem with the Same Solution
IN: [ 2, 3, 6, 8, 25, 50 ] TGT: 202

[ 2, 3, 6, 8, 25, 50 ] -> 8 * 25 = 200 -> [ 2, 3, 6, 50, 200 ] ;
BACK -> [ 2, 3, 6, 8, 25, 50 ] ;
[ 2, 3, 6, 8, 25, 50 ] -> 6 * 25 = 150 -> [ 2, 3, 8, 50, 150 ] ;
BACK -> [ 2, 3, 6, 8, 25, 50 ] ;
[ 2, 3, 6, 8, 25, 50 ] -> 6 * 50 = 300 -> [ 2, 3, 8, 25, 300 ] ;
BACK -> [ 2, 3, 6, 8, 25, 50 ] ;
[ 2, 3, 6, 8, 25, 50 ] -> 8 * 25 = 200 -> [ 2, 3, 6, 50, 200 ] ;
[ 2, 3, 6, 50, 200 ] -> 2 + 200 = 202 -> [ 3, 6, 50, 202 ] ;
SUCCESS ;
EXPR ((2 + (8 * 25))) ✔
```

C ADDITIONAL PROBING PLOT RESULT

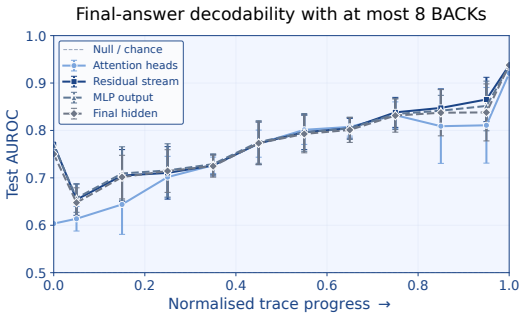


Figure 9: As in Figure 2, final-answer correctness becomes more decodable as generation progresses, even when we restrict traces to at most eight BACK transitions. This suggests that the increase is not primarily driven by traces with unusually frequent backtracking. Error bars show one standard deviation around the mean.

D MODEL TRAINING

We train the model using *standard token-level cross-entropy* over the trace, with the problem specification masked from the loss. The trajectories themselves provide supervision for arithmetic and search behaviour (without additional losses that explicitly reward correct operations or other desired behaviours).

We use digit-wise tokenisation, representing `899` as `8 9 9`, so that a fixed vocabulary can express the larger values produced by intermediate calculations. For correctness direction analysis, the last token position within a given number is considered as representative of the entire number when varying the semantic token position p (introduced in Section 3.1).

In preliminary experiments, we trained the model directly on traces generated using these heuristics within a modified A* algorithm (see Appendix B.2). However, the model's generated traces contained arithmetic errors, inconsistent states, and repetitive arithmetic operations despite stable training loss. We therefore include a pretraining phase, where the model is exposed to randomly generated exploratory traces containing valid arithmetic operations, state transitions, and backtracking, without requiring that they reach the target. This stage is intended to familiarise the model with valid trace structure before subsequent fine-tuning on A* solution traces.

How do we separate training and evaluation when distinct problems can share the same solution?

Because the task does not require using every input, distinct problems can share the same solution and search structure. For example, the inputs "`IN: [1,2,6,8,25,50] TGT: 202`" and "`IN: [2,3,6,8,25,50] TGT: 202`", both admit the same solution of `EXPR (2+(8*25))`. Furthermore, their corresponding reasoning traces would also be highly similar (besides the single number swap) as illustrated in Example 2 in Appendix B.3, requiring a similar sequence of arithmetic operations and `BACK` transitions.

Table 3: Target-disjoint splitting is motivated by the observation that even consecutive targets can lead to substantially different solution structures for a fixed input, here `IN: [1,8,8,9,9,100]`.

Target	Final `EXPR`	Steps
101	`(100+1)`	1
102	`(100+(9-(8-1)))`	3
103	`(100+(9-((8-(9-8))-1)))`	5
104	`(100+(8/(9-(8-1))))`	4

Target-Disjoint Splitting: We address this by assigning all problems with the same target to the same split, reducing the risk that closely related solution traces appear in both training and evaluation. Changing the target, even between consecutive integers, can lead to substantially different solutions while the inputs remain fixed (see Table 3).

Model Competence: Table 4 reports how reliably our model generates valid arithmetic operations, consistent state transitions, and correct final expressions after each training stage.

Table 4: The model achieves high arithmetic operation validity and state transition consistency, suggesting its competence. A final-answer accuracy of (71%) provides a useful middle ground where performance is strong, while the remaining (29%) of traces provide enough negative examples to limit class imbalance during probe training and CAA construction. Pretraining results use 504 validation examples, and fine-tuning results use 9,000 test examples (matching the corresponding data splits outlined in Appendix B.3 Table 2).

Metric	*What it measures*	*After pretraining*	*After fine-tuning*
Expression parseability	The final `EXPR` can be parsed	95.63%	99.33%
Arithmetic operation validity	A generated arithmetic operation is legal	93.58%	96.37%
State consistency	The resulting state from a state transition correctly reflects the arithmetic operation	92.93%	95.16%
Final-answer correctness	The final `EXPR` evaluates to the target	25.20%	71.12%

E TYPES OF LINEAR STEERING

We differ from prior work by focusing specifically on whether reasoning correctness itself can be steered. Venhoff et al. (2025) instead steer higher-level reasoning behaviours such as uncertainty, example testing, and backtracking. Chen et al. (2025b) and Zhuang et al. (2026) show that steering behaviours such as reflection and transition can indirectly improve answer accuracy, but the underlying directions encode those behaviours rather than correctness itself. Four constructions are especially relevant, as summarized in Table 5.

Table 5: Four constructions for linear intervention directions. Here, $h_{i,p,c}$ denotes the activation for example i at token position p and model component c, $y_i \in \{0, 1\}$ denotes the corresponding binary label, and θ_{ft} and θ_{pre} denote the parameters of the finetuned and pretrained model respectively. Here, M denotes the frozen model, v denotes the direction produced by each construction, b denotes the learned probe bias, with its joint optimization omitted from the notation for brevity, a_i denotes the target answer-token sequence for example i, and $\mathcal{L}$ denotes the classification loss for the probe, and the objective for direct optimisation direction, respectively.

	Parameter-space task vector	**Probe direction**	**Mean-difference activation direction (CAA)**	**Directly optimized activation direction**
Construction	$v = \theta_{ft} - \theta_{pre}$	$v_{p,c} = \arg\min_v \sum_i \{ \mathcal{L}(y_i, v^\top h_{i,p,c} + b)\}$	$v_{p,c} = \mathbb{E}[h_{i,p,c} \mid y_i = 1] - \mathbb{E}[h_{i,p,c} \mid y_i = 0]$	$v_{p,c} = \arg\min_v \sum_i \{ \mathcal{L}\left(M(h_{i,p,c} + v), a_i\right)\}$
Space	Parameter space	Activation space	Activation space	Activation space

Our causal experiments compare the three activation-space direction vector methods, but the parameter-space task vector method is a nice contrast:

I. *Parameter-space task vectors*: Task vectors measure the parameter difference between two models, typically reflecting the changes introduced by task adaptation (Ilharco et al., 2023). *We do not use this approach*, and instead focus on the activation-space constructions in Items II., III., and IV.. This is because the parameter difference is not conditioned on correctness and may therefore capture broader task-related changes rather than correctness specifically. In particular, the direction vector $v = \theta_{ft} - \theta_{pre}$ is not defined at a specific token position p or model component c, and is therefore not denoted $v_{p,c}$ like the activation-space methods.

II. *Probe-based steering*: A linear probe learns a discriminative direction that separates activations associated with the two classes. The learned probing weight vector $v_{p,c}$ can then be used directly as an intervention direction, as in prior work on properties such as truthfulness and sycophancy (Li et al., 2023; Genadi et al., 2026).

III. *Mean-difference activation steering*: Contrastive Activation Addition constructs the steering direction $v_{p,c}$ from the displacement between the mean activations of two classes (Rimsky et al., 2024). Related approaches have been used to steer truthfulness and other high-level behaviours (Li et al., 2023; Rimsky et al., 2024), as well as reasoning behaviours including uncertainty, backtracking, reflection, and transition (Venhoff et al., 2025; Chen et al., 2025b; Zhuang et al., 2026). We compute these directions at the same semantic positions and model components as the linear probes. This gives a direct comparison between a discriminatively learned direction (in II above) and the corresponding difference between class means.

IV. *Directly optimized activation steering*: Unlike probe-based and mean-difference steering, this approach does not derive its direction from a classifier or class-conditional activation statistics. Instead, it backpropagates through the frozen model while updating only an additive activation-space vector $v_{p,c}$ (Cao et al., 2024; Dunefsky & Cohan, 2025). In our setting, a reusable vector is optimized over many training examples and their target answers a_i, for a given semantic position and model component. This provides a control for distinguishing failures of direction construction from broader limits of local additive steering.

F Direct Steering Vector Optimisation Details

As discussed in Section 3.1 and Appendix Section E, Direct Optimisation provides an optimisation-based control experiment for comparison with probe- and CAA-based steering.

F.1 Direct Steering Vector Optimisation Objective

For Direct Optimisation, we freeze the model M and optimise only the additive vector,

$$v_{p,c}^{\text{opt}} = \arg\min_{v} \sum_{i} \mathcal{L}_{\text{ans}} \left(M(h_{i,p,c} + v), a_i \right),$$

where a_i is the selected valid final-answer token sequence. *When the model's generated expression is correct, we use that expression as a_i. Otherwise, we use the corresponding A^* reference expression.* This avoids penalising valid expressions that differ syntactically from the reference.

The loss is computed on a_i conditioned on the model's original generated prefix,

$$\mathcal{L}_{\text{ans}} \left(M(h_{i,p,c} + v), a_i \right) = -\frac{1}{|a_i|} \sum_{t=1}^{|a_i|} \log p_M \left(a_{i,t} \mid a_{i,<t}, h_{i,p,c} + v \right),$$

where the fixed preceding prefix is omitted from the conditioning notation. Thus, optimising v maximises the mean log-probability assigned to the selected valid final-answer tokens. We select the vector with the lowest loss on final-answer tokens in the validation data, then evaluate whether it improves final-answer correctness when the model generates a new response in the later steering experiments. Where multiple selected token positions from the same trace can contribute, examples are weighted so that each trace contributes equally, and we apply a small L_2 penalty to v.

F.2 Optimisation Convergence

During Direct Optimisation, we evaluate the vector every 20 optimisation steps. At each validation point t, we measure the cosine similarity between the current vector and the vector from the preceding validation point, $s_t = \cos(v_t, v_{t-20})$. We treat a run as directionally stable when both the final value of s_t and its median over the final four validation intervals, corresponding to the last 80 optimisation steps, are at least 0.999. Values close to 1 indicate that the vector's orientation has stabilised even if its norm or validation loss continues to change.

F.3 Supervision Signal

Reachability has no equally direct objective because M never predicts whether a state is reachable. One option is to introduce a reachability classifier, but then Direct Optimisation would depend on a probe. Another option is to let v affect subsequent generation and supervise whether the resulting states remain reachable. Because reachability can change at every step, this becomes repeated supervision over the evolving trace rather than optimisation of a single local reachability direction.

We therefore use final-answer supervision for Direct Optimisation, while the probe and CAA constructions retain both reachability and final-answer labels. `SUCCESS` remains eligible because `EXPR` follows it, while the final parenthesis of `EXPR` is excluded because the answer has already been produced.

F.4 Gradient-Path Constraints

Backpropagation imposes a further constraint because Direct Optimisation requires a differentiable path from v to the logits for later answer tokens. In our four-layer model, interventions at layers 0–2 can still influence later tokens through subsequent attention layers, so $\partial \mathcal{L}_{\text{ans}} / \partial v \neq 0$. At layer 3 and the final hidden representation, no subsequent attention layer remains to carry the perturbation forward, giving $\partial \mathcal{L}_{\text{ans}} / \partial v = 0$. We therefore exclude these locations from Direct Optimisation, leaving 30 eligible model locations per semantic position rather than the 41 used for probe and CAA vectors. Probe and CAA vectors from the excluded locations are retained because they can still affect free generation by changing the next emitted token, even though this discrete route cannot carry gradients.

G Additional Correctness Direction Results

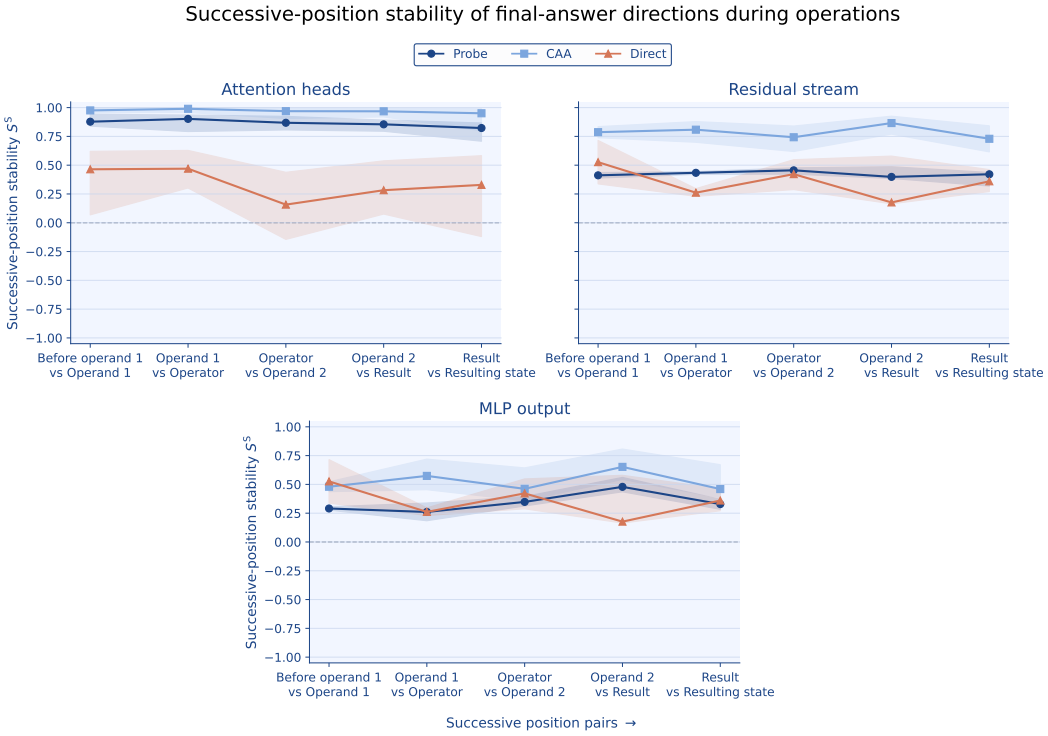


Figure 10: We measure successive-position stability (S^{S}) to examine whether **final-answer** directions remain similar as the model constructs an arithmetic state transition. Probe and CAA final-answer directions show high stability at attention heads, remaining similar across successive token positions. Directions obtained through Direct Optimisation show lower stability, while residual-stream and MLP directions are less stable or vary more in stability across positions. Lines show medians across locations within each component class, and shading shows interquartile ranges.

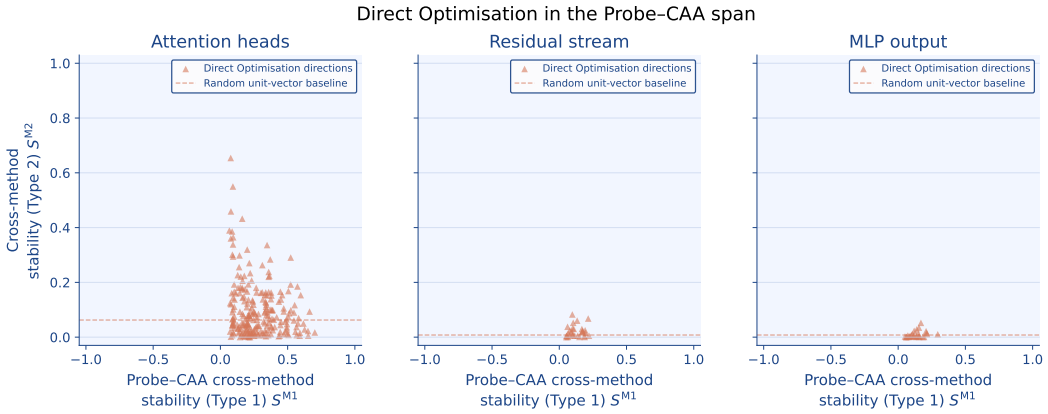


Figure 11: Plotting cross-method stability Type 2 (S^{M2}) against Probe–CAA cross-method stability Type 1 (S^{M1}) reveals that the strongest agreement with Direct Optimisation is concentrated at a small subset of attention-head locations. S^{M2} measures how much of a Direct Optimisation direction lies within the plane spanned by Probe and CAA, even when its cosine similarity with either vector individually is low. Some attention-head directions have substantial projections onto this plane, while projections at residual streams and MLP outputs remain close to the random baseline. Each point compares directions at the same token position and model location. Dashed lines show median projections of random unit vectors.

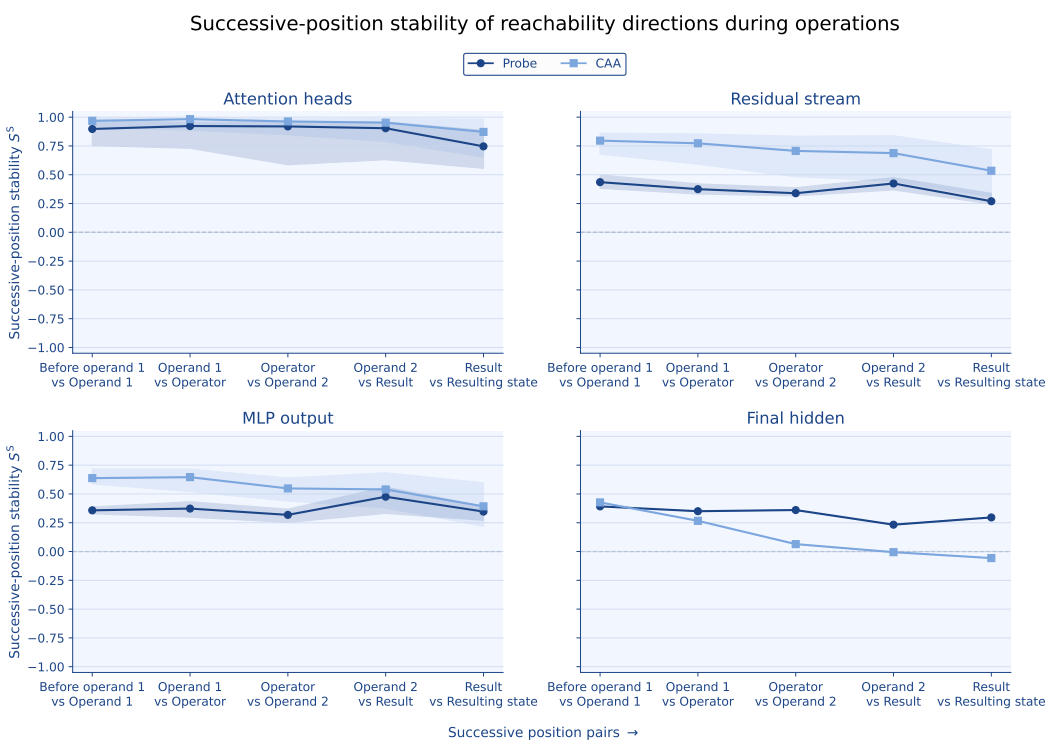


Figure 12: We measure successive-position stability (S^{S}) to examine how **reachability** directions change as an arithmetic state transition is constructed. Probe and CAA reachability directions remain similar across successive token positions at attention heads, with lower stability in other model components. Successive-position stability for CAA directions at the final hidden representation approaches zero at later transitions. Lines show medians across locations within each component class, and shading shows interquartile ranges.

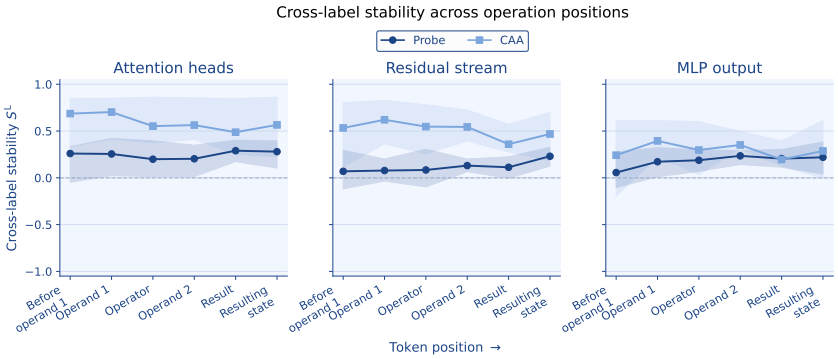


Figure 13: Cross-label stability (S^{L}) measures whether next-state reachability and final-answer correctness are associated with similar directions at the same token position and model location. CAA generally yields higher cross-label stability than Probe, particularly at attention heads and the residual stream, although the two correctness labels do not share an identical direction. Lines show median cosine similarity within each component class, and shading shows interquartile ranges.

H Additional Causal Steering Analyses

H.1 Additional Attention-Head Interventions

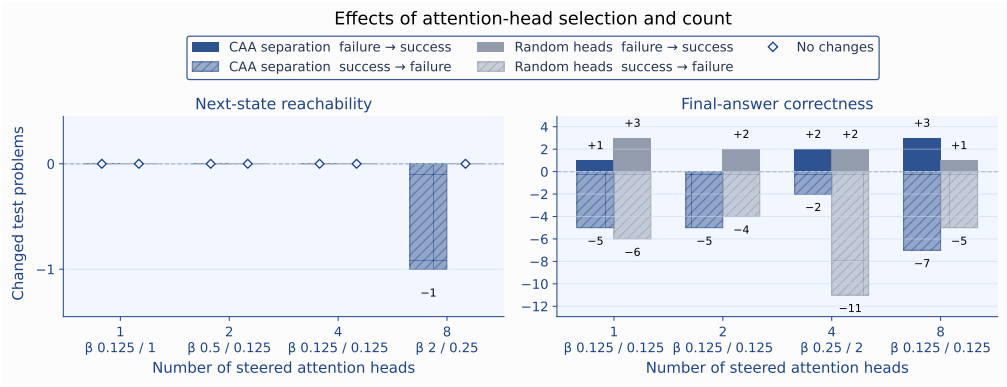


Figure 14: We vary the number of steered heads and compare selection by CAA separation with random selection, measuring changes in next-state reachability and final-answer correctness. Below each head count, the two β values are chosen using validation data and shown as "`CAA selection / random selection`". Each setting is evaluated on 512 test problems, with and without steering. Selecting heads by CAA separation gives no consistent advantage over random selection.

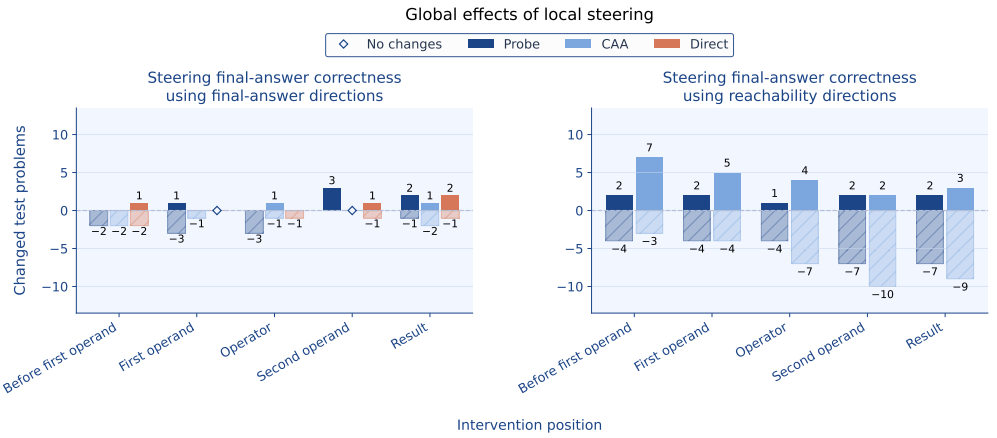


Figure 15: We compare whether final-answer correctness improves when steering uses directions constructed for final-answer correctness (left) or next-state reachability (right). Neither type of direction produces consistently beneficial effects across intervention positions. All interventions use four heads, with steering strengths chosen separately using validation data. Each setting is evaluated on 512 test problems, with and without steering.

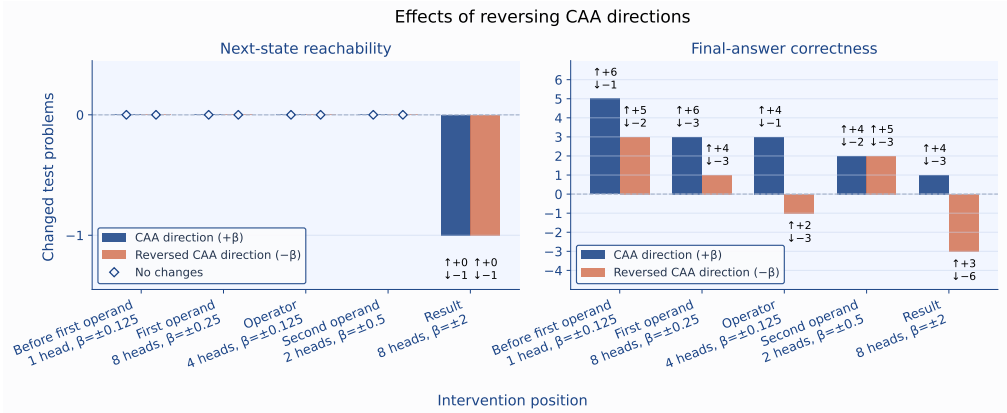


Figure 16: We reverse the CAA sign chosen using validation data to test whether the opposite direction reverses its effect on next-state reachability and final-answer correctness. Bar heights show the number of problems improved minus the number worsened, with annotations reporting both counts. Neither sign consistently outperforms the other. Each sign is evaluated on the same 512 test problems and compared with generation without steering.

H.2 Broader Steering Interventions Beyond Attention Heads

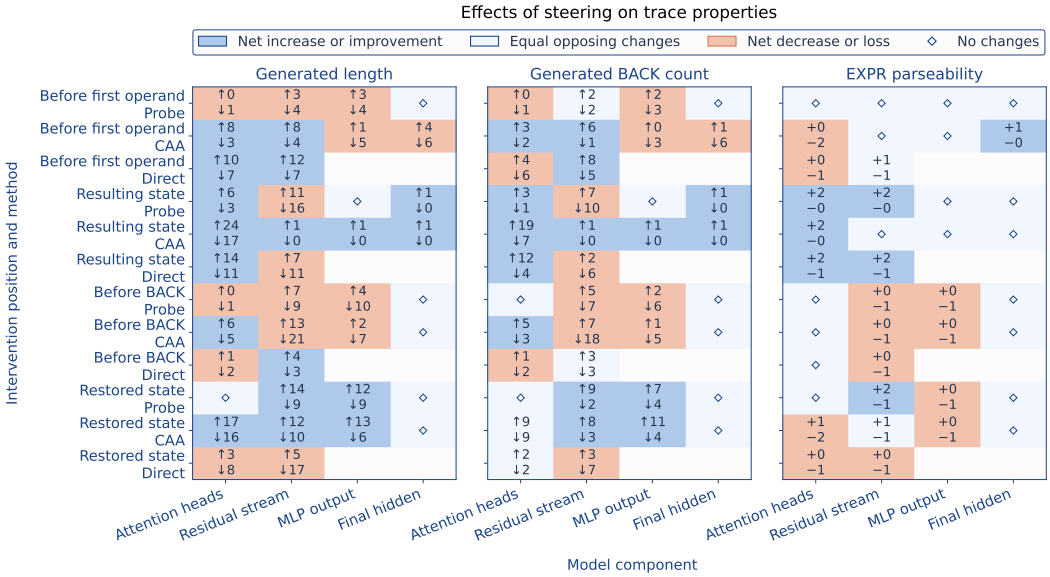
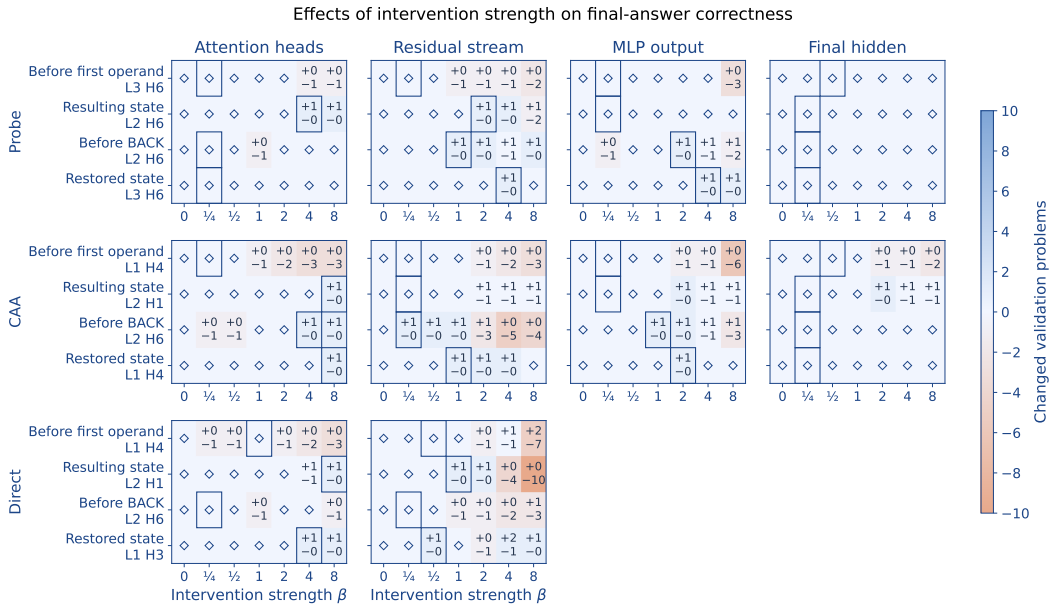
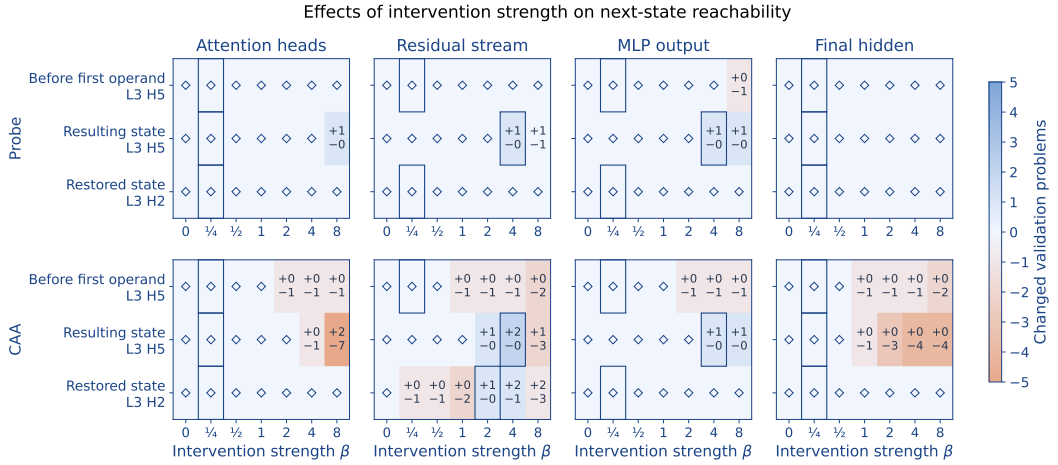


Figure 17: We measure changes in generation length, `BACK` usage, and expression parseability to examine how steering affects the generated trace beyond final-answer correctness. These effects vary across methods, model components, and intervention positions, without a common pattern of improved reasoning. The first two panels count problems with increases and decreases separately, while the third shows the number gaining a parseable expression minus the number losing one. Each setting is evaluated on 256 test problems, with and without steering.



(a)



(b)

Figure 18: Varying steering strength β does not reveal a consistently beneficial response in either final-answer correctness (a) or next-state reachability (b). Larger interventions often change more outcomes, but some are improved and others worsened. Each cell reports the numbers of problems improved and worsened, and outlined cells mark strengths chosen using validation data. Each setting is evaluated on 16 validation problems, with and without steering.

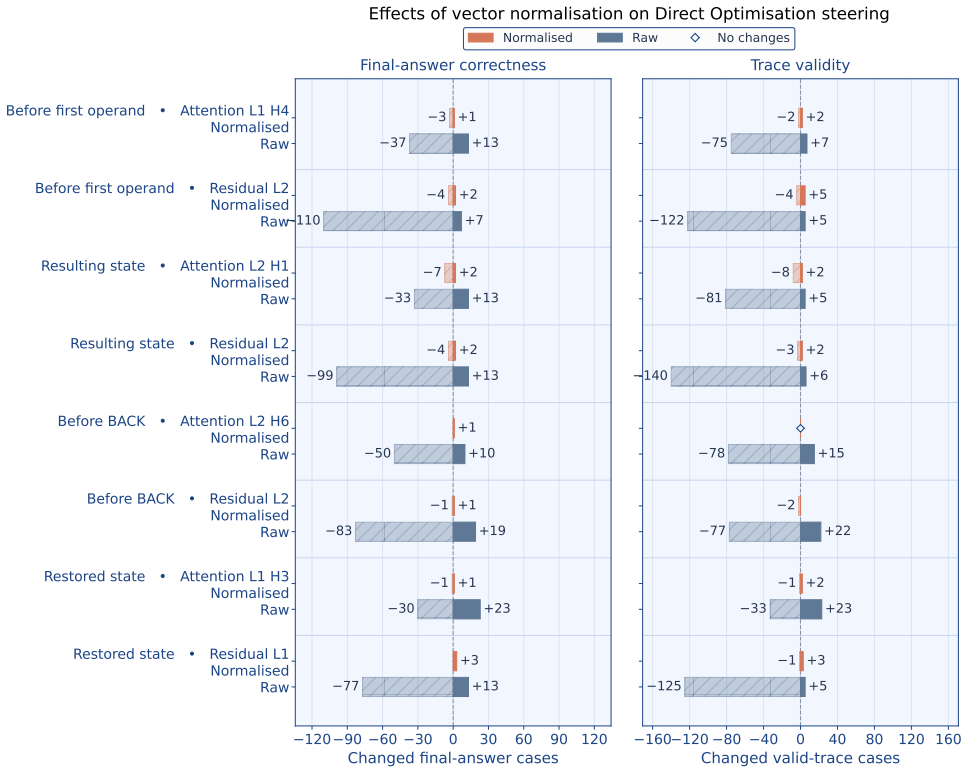


Figure 19: Applying the vector learned through Direct Optimisation without normalising it changes more outcomes than applying its version normalised to unit length, but more outcomes are worsened than improved for both final-answer correctness and trace validity in these settings. We compare the two versions at the same intervention position and model component. Each version is evaluated on the same 256 test problems, with and without steering.

I FULL LIST OF HEURISTICS

1. Target-Driven Lookahead

This family of heuristics begins with the target number, working backward to decompose the problem or prune the search space before any calculations are made.

1a. The Difference Method

Strategy Description: This is the dominant procedural pattern, embodying a 'Multiply Early, Adjust Late' game plan. A player performs a large multiplication to create an anchor number close to the target, then reframes the problem into solving for the small remaining difference (Countdown Wiki, 2024; Defays, 1995; DataGenetics, 2014).

Potential Reward Signal: A positive reward proportional to the reduction in absolute distance between an intermediate result and the final target.

1b. Quotient Foresight

Strategy Description: A related lookahead tactic where players mentally check if the target is roughly a multiple of an available number. They assess whether the integer quotient '`(Target / x)`' is easily constructible from the remaining numbers before committing to a path (Defays, 1995; Countdown Wiki, 2024).

Potential Reward Signal: A bonus for creating a number that is an exact quotient, or for moving into a state where the new subproblem is solvable.

1c. Residue & Parity Pruning

Strategy Description: An advanced tactic where players use number properties to prune impossible solution paths. Instead of calculating a full result, they use fast mental shortcuts to check for plausibility. Common techniques include:

- Parity (Odd/Even): If the target is odd, the final arithmetic operation cannot be even * anything, even + even, or even - even, eliminating many branches.
- Last Digit (mod 10): The target's final digit constrains the final operation. A target ending in 3' guides the search towards combinations that can produce a 3.
- Large Number Residue (mod 25): For a target like 783, a player might recognize 783 = 25 * 31 + 8, reframing the problem into a subgoal of making the remainder 8.

These checks allow a player to quickly discard entire subtrees of the search space without performing the full calculations (Resources, 2024).

Potential Reward Signal: A multi-component bonus awarded for actions that align with the target's properties:

- Parity Match: A small bonus if an intermediate result has the same parity (odd/even) as the final target.
- Last Digit Match: A small bonus if an arithmetic operation produces a result whose last digit matches the target's last digit.
- Remainder Subgoal: A more significant bonus for creating an intermediate number that is exactly equal to the target's remainder when divided by a large number (25, 50, or 75) that is currently in the state.

2. Structural Decomposition

Rather than a linear path, these strategies involve breaking the problem itself into more manageable, independent parts.

2a. Decomposing the Target (Factorization)

Strategy Description: A powerful divide-and-conquer tactic where the target is mentally decomposed into its factors (e.g., `744 = 8 x 93`). These factors then become simpler, intermediate sub-goals to be solved (Countdown Wiki, 2024; Defays, 1995).

Potential Reward Signal: A positive reward for creating an intermediate number that is a nontrivial factor of the target number.

2b. Decomposing the Numbers (Partition-and-Combine)

Strategy Description: A structural pattern where the player mentally splits the available numbers into two groups, solves for two intermediate subresults, and then combines them with a final operation, such as '(a+b) * (c+d)' (Alliot & Vanaret, 2015; Defays, 1995).

Potential Reward Signal: A positive reward if two separate intermediate results in the state can be combined to directly reach the target.

3. Constructive Planning & Waypoint Creation

This family of heuristics covers the idea that expert players don't just use the numbers they are given; they actively construct the numbers they need.

3a. Building Operands (Legal Decomposition)

Strategy Description: A constructive tactic where players build a required operand on the fly. This is a legal application of the distributive property, such as forming a multiplier by first summing two smaller numbers, e.g., finding `x * (a + b)`. This demonstrates multi-level planning, where one calculation serves as a subroutine for another (Countdown Wiki, 2024).

Potential Reward Signal: A positive reward for using the result of one arithmetic operation as an operand in an immediate subsequent multiplication.

3b. Building Waypoints (High-Divisor Composites)

Strategy Description: A broader strategy where players deliberately form cognitively easy and mathematically flexible waypoints. Expert players learn to prefer creating "high-divisor composites" (e.g., 60, 72, 90) over large primes, as these numbers act as powerful hubs with many factors for subsequent steps (Resources, 2024; Defays, 1995; DataGenetics, 2014).

Potential Reward Signal: A small bonus for hitting waypoints from a predefined list of "useful composites," especially if they reduce the distance to the target.

3c. Strategic Doubling and Halving

Strategy Description: Players treat certain numbers as functional tools for rapid scaling. Numbers like 2 and 4 are used for power-of-two scaling to quickly approach the target's order of magnitude. A common application is the "half-difference trick," where a player creates a state where the remaining gap to the target can be easily bridged by a final doubling or halving calculation (Countdown Wiki, 2024).

Potential Reward Signal: A small positive reward for any multiplication or division by 2 or 4 that reduces the distance to the target.

4. Strategic Foresight & Resource Management

These advanced strategies require looking ahead to evaluate moves based on their future consequences.

4a. Preserving High-Value Pairs (Partner Preservation)

Strategy Description: A sophisticated strategy of preserving "canonical partners" for high-impact moves (e.g., saving a `4` to be used with a 25). This requires evaluating moves based on their opportunity cost (Countdown Wiki, 2024; Resources, 2024).

Potential Reward Signal: A small negative reward (cost) for using a canonical partner in a low-value arithmetic operation (e.g., addition).

4b. Maintaining State Flexibility (Composite Preference)

Strategy Description: A key aspect of strategic foresight is maintaining a flexible board state by preferring to create intermediate numbers that are themselves easy to decompose. Expert players develop an intuitive preference for "high-divisor composites" (numbers with many small, obvious factors, e.g., 60, 72, 90) as these act as powerful and flexible hubs for subsequent arithmetic operations. Conversely, this leads to an implicit strategy of "Early Prime-Avoidance" (Resources, 2024).

Potential Reward Signal: A small negative reward (cost) for generating a large intermediate number that is not divisible by 2, 3, or 5, unless it is already very close to the target.

5. Meta-Gaming & Efficiency

Finally, expert play is shaped by the rules of the game and the cognitive limits of the player.

5a. Adapting to Scoring Rules (Near-Miss Strategy)

Strategy Description: A pragmatic shift from an optimization objective (find the exact target) to a satisficing one (land within the `±10` scoring range) when an exact solution seems intractable under time pressure (Wikipedia contributors, 2025; Countdown Wiki, 2024).

Potential Reward Signal: In a multi-objective setting, a secondary reward for terminating within the `±10` scoring range.

5b. Optimizing for Simplicity (Step & Identity Efficiency)

Strategy Description: Skilled players prefer shorter solutions and avoid “wasted” moves that offer no progress, such as adding or subtracting zero, or multiplying or dividing by one. This cognitive preference for efficiency is also a key metric for optimality in algorithmic solvers (Alliot & Vanaret, 2015; DataGenetics, 2014).

Potential Reward Signal: A small, per-step penalty to encourage efficiency, and a larger penalty for generating a useless identity calculation.

J ADDITIONAL CHAINSEARCH TRACE EXAMPLES

The examples below provide one SFT sample reasoning trace per number of BACK transitions, from one through eight. The traces are reference trajectories rather than unique solutions, as a problem may admit multiple valid traces.

Example 3. An A* reference trace with **one** BACK transitions

```
Problem
IN: [ 1, 1, 2, 9, 25, 50 ] TGT: 675

Solution
[ 1, 1, 2, 9, 25, 50 ] -> 9 * 50 = 450
-> [ 1, 1, 2, 25, 450 ] ;
BACK -> [ 1, 1, 2, 9, 25, 50 ] ;
[ 1, 1, 2, 9, 25, 50 ] -> 25 + 50 = 75
-> [ 1, 1, 2, 9, 75 ] ;
[ 1, 1, 2, 9, 75 ] -> 9 * 75 = 675
-> [ 1, 1, 2, 675 ] ;
SUCCESS ;
EXPR ((9 * (25 + 50))) ✔
```

Example 4. An A* reference trace with **two** BACK transitions

```
Problem
IN: [ 3, 3, 9, 9, 25, 50 ] TGT: 275

Solution
[ 3, 3, 9, 9, 25, 50 ] -> 3 * 50 = 150
-> [ 3, 9, 9, 25, 150 ] ;
BACK -> [ 3, 3, 9, 9, 25, 50 ] ;
[ 3, 3, 9, 9, 25, 50 ] -> 9 * 50 = 450
-> [ 3, 3, 9, 25, 450 ] ;
BACK -> [ 3, 3, 9, 9, 25, 50 ] ;
[ 3, 3, 9, 9, 25, 50 ] -> 9 * 25 = 225
-> [ 3, 3, 9, 50, 225 ] ;
[ 3, 3, 9, 50, 225 ] -> 50 + 225 = 275
-> [ 3, 3, 9, 275 ] ;
SUCCESS ;
EXPR ((50 + (9 * 25))) ✔
```

Example 5. An A* reference trace with **three** BACK transitions

```
Problem
IN: [ 5, 7, 7, 9, 25, 100 ] TGT: 108

Solution
[ 5, 7, 7, 9, 25, 100 ] -> 7 + 100 = 107
-> [ 5, 7, 9, 25, 107 ] ;
BACK -> [ 5, 7, 7, 9, 25, 100 ] ;
[ 5, 7, 7, 9, 25, 100 ] -> 5 + 100 = 105
-> [ 7, 7, 9, 25, 105 ] ;
BACK -> [ 5, 7, 7, 9, 25, 100 ] ;
[ 5, 7, 7, 9, 25, 100 ] -> 9 + 100 = 109
-> [ 5, 7, 7, 25, 109 ] ;
BACK -> [ 5, 7, 7, 9, 25, 100 ] ;
[ 5, 7, 7, 9, 25, 100 ] -> 5 + 7 = 12
-> [ 7, 9, 12, 25, 100 ] ;
[ 7, 9, 12, 25, 100 ] -> 9 * 12 = 108
-> [ 7, 25, 100, 108 ] ;
SUCCESS ;
EXPR ((9 * (5 + 7))) ✔
```

Example 6. An A* reference trace with **four** BACK transitions

```
Problem
IN: [ 1, 5, 9, 9, 25, 75 ] TGT: 120

Solution
[ 1, 5, 9, 9, 25, 75 ] -> 25 + 75 = 100
-> [ 1, 5, 9, 9, 100 ] ;
BACK -> [ 1, 5, 9, 9, 25, 75 ] ;
[ 1, 5, 9, 9, 25, 75 ] -> 9 + 75 = 84
-> [ 1, 5, 9, 25, 84 ] ;
BACK -> [ 1, 5, 9, 9, 25, 75 ] ;
[ 1, 5, 9, 9, 25, 75 ] -> 5 * 25 = 125
-> [ 1, 9, 9, 75, 125 ] ;
BACK -> [ 1, 5, 9, 9, 25, 75 ] ;
[ 1, 5, 9, 9, 25, 75 ] -> 5 + 75 = 80
-> [ 1, 9, 9, 25, 80 ] ;
BACK -> [ 1, 5, 9, 9, 25, 75 ] ;
[ 1, 5, 9, 9, 25, 75 ] -> 25 - 1 = 24
-> [ 5, 9, 9, 24, 75 ] ;
[ 5, 9, 9, 24, 75 ] -> 5 * 24 = 120
-> [ 9, 9, 75, 120 ] ;
SUCCESS ;
EXPR ((5 * (25 - 1))) ✔
```

Example 7. An A* reference trace with **five** BACK transitions

```
Problem
IN: [ 2, 3, 3, 5, 9, 50 ] TGT: 135

Solution
[ 2, 3, 3, 5, 9, 50 ] -> 3 * 50 = 150
-> [ 2, 3, 5, 9, 150 ] ;
BACK -> [ 2, 3, 3, 5, 9, 50 ] ;
[ 2, 3, 3, 5, 9, 50 ] -> 2 * 50 = 100
-> [ 3, 3, 5, 9, 100 ] ;
BACK -> [ 2, 3, 3, 5, 9, 50 ] ;
[ 2, 3, 3, 5, 9, 50 ] -> 3 * 50 = 150
-> [ 2, 3, 5, 9, 150 ] ;
[ 2, 3, 5, 9, 150 ] -> 150 - 9 = 141
-> [ 2, 3, 5, 141 ] ;
BACK -> [ 2, 3, 5, 9, 150 ] ;
[ 2, 3, 5, 9, 150 ] -> 150 - 5 = 145
-> [ 2, 3, 9, 145 ] ;
BACK -> [ 2, 3, 5, 9, 150 ] ;
[ 2, 3, 5, 9, 150 ] -> 150 - 3 = 147
-> [ 2, 5, 9, 147 ] ;
BACK -> [ 2, 3, 3, 5, 9, 50 ] ;
[ 2, 3, 3, 5, 9, 50 ] -> 5 * 9 = 45
-> [ 2, 3, 3, 45, 50 ] ;
[ 2, 3, 3, 45, 50 ] -> 3 * 45 = 135
-> [ 2, 3, 50, 135 ] ;
SUCCESS ;
EXPR ((3 * (5 * 9))) ✔
```

Example 8. An A* reference trace with **six** BACK transitions

```
Problem
IN: [ 3, 7, 7, 8, 10, 75 ] TGT: 248

Solution
[ 3, 7, 7, 8, 10, 75 ] -> 3 * 75 = 225
-> [ 7, 7, 8, 10, 225 ] ;
BACK -> [ 3, 7, 7, 8, 10, 75 ] ;
[ 3, 7, 7, 8, 10, 75 ] -> 3 + 75 = 78
```

```
-> [ 7, 7, 8, 10, 78 ] ;
BACK -> [ 3, 7, 7, 8, 10, 75 ] ;
[ 3, 7, 7, 8, 10, 75 ] -> 3 * 75 = 225
-> [ 7, 7, 8, 10, 225 ] ;
[ 7, 7, 8, 10, 225 ] -> 10 + 225 = 235
-> [ 7, 7, 8, 235 ] ;
BACK -> [ 7, 7, 8, 10, 225 ] ;
[ 7, 7, 8, 10, 225 ] -> 7 + 225 = 232
-> [ 7, 8, 10, 232 ] ;
BACK -> [ 3, 7, 7, 8, 10, 75 ] ;
[ 3, 7, 7, 8, 10, 75 ] -> 8 * 10 = 80
-> [ 3, 7, 7, 75, 80 ] ;
[ 3, 7, 7, 75, 80 ] -> 3 * 80 = 240
-> [ 7, 7, 75, 240 ] ;
BACK -> [ 3, 7, 7, 8, 10, 75 ] ;
[ 3, 7, 7, 8, 10, 75 ] -> 3 * 75 = 225
-> [ 7, 7, 8, 10, 225 ] ;
[ 7, 7, 8, 10, 225 ] -> 8 + 225 = 233
-> [ 7, 7, 10, 233 ] ;
BACK -> [ 3, 7, 7, 8, 10, 75 ] ;
[ 3, 7, 7, 8, 10, 75 ] -> 10 + 75 = 85
-> [ 3, 7, 7, 8, 85 ] ;
[ 3, 7, 7, 8, 85 ] -> 3 * 85 = 255
-> [ 7, 7, 8, 255 ] ;
[ 7, 7, 8, 255 ] -> 255 - 7 = 248
-> [ 7, 8, 248 ] ;
SUCCESS ;
EXPR (((3 * (10 + 75)) - 7)) ✔
```

Example 9. An A* reference trace with **seven** BACK transitions

```
Problem
IN: [ 1, 2, 5, 7, 8, 75 ] TGT: 515

Solution
[ 1, 2, 5, 7, 8, 75 ] -> 7 * 75 = 525
-> [ 1, 2, 5, 8, 525 ] ;
BACK -> [ 1, 2, 5, 7, 8, 75 ] ;
[ 1, 2, 5, 7, 8, 75 ] -> 8 * 75 = 600
-> [ 1, 2, 5, 7, 600 ] ;
BACK -> [ 1, 2, 5, 7, 8, 75 ] ;
[ 1, 2, 5, 7, 8, 75 ] -> 2 * 75 = 150
-> [ 1, 5, 7, 8, 150 ] ;
BACK -> [ 1, 2, 5, 7, 8, 75 ] ;
[ 1, 2, 5, 7, 8, 75 ] -> 5 * 75 = 375
-> [ 1, 2, 7, 8, 375 ] ;
BACK -> [ 1, 2, 5, 7, 8, 75 ] ;
[ 1, 2, 5, 7, 8, 75 ] -> 7 * 75 = 525
-> [ 1, 2, 5, 8, 525 ] ;
[ 1, 2, 5, 8, 525 ] -> 525 - 8 = 517
-> [ 1, 2, 5, 517 ] ;
BACK -> [ 1, 2, 5, 7, 8, 75 ] ;
[ 1, 2, 5, 7, 8, 75 ] -> 5 + 75 = 80
-> [ 1, 2, 7, 8, 80 ] ;
BACK -> [ 1, 2, 5, 7, 8, 75 ] ;
[ 1, 2, 5, 7, 8, 75 ] -> 7 * 75 = 525
-> [ 1, 2, 5, 8, 525 ] ;
[ 1, 2, 5, 8, 525 ] -> 525 - 2 = 523
-> [ 1, 5, 8, 523 ] ;
BACK -> [ 1, 2, 5, 8, 525 ] ;
[ 1, 2, 5, 8, 525 ] -> 525 - 8 = 517
-> [ 1, 2, 5, 517 ] ;
[ 1, 2, 5, 517 ] -> 517 - 2 = 515
-> [ 1, 5, 515 ] ;
SUCCESS ;
EXPR ((((7 * 75) - 8) - 2)) ✔
```

Example 10. An A* reference trace with **eight** BACK transitions

```
Problem
IN: [ 4, 6, 7, 9, 75, 100 ] TGT: 193

Solution
[ 4, 6, 7, 9, 75, 100 ] -> 75 + 100 = 175
-> [ 4, 6, 7, 9, 175 ] ;
[ 4, 6, 7, 9, 175 ] -> 9 + 175 = 184
-> [ 4, 6, 7, 184 ] ;
BACK -> [ 4, 6, 7, 9, 175 ] ;
[ 4, 6, 7, 9, 175 ] -> 7 + 175 = 182
-> [ 4, 6, 9, 182 ] ;
BACK -> [ 4, 6, 7, 9, 175 ] ;
[ 4, 6, 7, 9, 175 ] -> 6 + 175 = 181
-> [ 4, 7, 9, 181 ] ;
BACK -> [ 4, 6, 7, 9, 175 ] ;
[ 4, 6, 7, 9, 175 ] -> 4 + 175 = 179
-> [ 6, 7, 9, 179 ] ;
BACK -> [ 4, 6, 7, 9, 75, 100 ] ;
[ 4, 6, 7, 9, 75, 100 ] -> 9 + 100 = 109
-> [ 4, 6, 7, 75, 109 ] ;
BACK -> [ 4, 6, 7, 9, 75, 100 ] ;
[ 4, 6, 7, 9, 75, 100 ] -> 7 + 100 = 107
-> [ 4, 6, 9, 75, 107 ] ;
BACK -> [ 4, 6, 7, 9, 75, 100 ] ;
[ 4, 6, 7, 9, 75, 100 ] -> 6 + 100 = 106
-> [ 4, 7, 9, 75, 106 ] ;
[ 4, 7, 9, 75, 106 ] -> 4 + 106 = 110
-> [ 7, 9, 75, 110 ] ;
BACK -> [ 4, 6, 7, 9, 75, 100 ] ;
[ 4, 6, 7, 9, 75, 100 ] -> 4 + 100 = 104
-> [ 6, 7, 9, 75, 104 ] ;
BACK -> [ 4, 6, 7, 9, 75, 100 ] ;
[ 4, 6, 7, 9, 75, 100 ] -> 4 * 75 = 300
-> [ 6, 7, 9, 100, 300 ] ;
[ 6, 7, 9, 100, 300 ] -> 300 - 100 = 200
-> [ 6, 7, 9, 200 ] ;
[ 6, 7, 9, 200 ] -> 200 - 7 = 193
-> [ 6, 9, 193 ] ;
SUCCESS ;
EXPR ((((4 * 75) - 100) - 7))
```