BEFORE REASONING BEGINS: TEMPORAL CONTRAST REVEALS A CAUSAL TARGET ACROSS REASONING TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Code training is one of the most effective empirical routes to improving language-model reasoning, yet existing evidence remains largely at the level of data-mixture correlations and continued-training gains. A central mechanistic challenge is to identify an internal target that can be localized and causally tested across reasoning tasks. Our key insight is that reasoning-related intervention targets can be discovered before reasoning is generated. For the same task content, chat template and raw continuation inputs occupy different final pre-generation states even though both may later produce reasoning traces. We use this functionally aligned contrast to screen sparse autoencoder features at the final prompt position, control for responses reproduced by isolated prompt components, and localize the earliest consistent trajectory separation to layers L23–L24. This pipeline yields a compact set of 23 intervention targets in Qwen3-8B. Applying targeted SAE-guided suppression to these features throughout generation across 2,187 examples from GSM8K, CruxEval-O, and HumanEvalPlus reduces sample-weighted accuracy from 86.1% to 41.2%, a drop of 44.9 percentage points. On GSM8K, same-size, layer-count-matched random null interventions reduce accuracy by only 4.4 points on average, compared with 44.2 points under the targeted intervention. Across the code benchmarks, targeted and random interventions exhibit task-dependent accuracy effects but distinct failure profiles. Visible chain-of-thought controls further show that removing visible reasoning does not reproduce the collapse, while forcing reasoning entry after intervention does not restore performance. These results identify a compact, high-leverage intervention target with broad causal influence across mathematical and code-related behavior. Together, these findings establish pre-generation contrast as a discovery strategy for identifying and testing causal targets across reasoning tasks. Our code is openly available at `https://anonymous.4open.science/r/anonymous-reasoning-core-CEF4/`.

1 INTRODUCTION

Why does training on code improve reasoning on tasks that contain no code? Evidence increasingly shows that greater code exposure is associated with stronger downstream reasoning, and that continued training on code can improve mathematical and logical capabilities even when these tasks are not the direct training objective (Petty et al., 2025; Ma et al., 2024; Kim et al., 2024). Yet the internal basis of this transfer remains unclear. This motivates a concrete mechanistic question: can we identify a compact internal target with causal influence across mathematical and code-related reasoning?

Finding such a target is difficult because reasoning preparation is a transient computation rather than a fixed semantic category. Common contrasts mix this state with changes in task content and difficulty, domain and token distribution, or expected output structure. Sparse autoencoders provide a manipulable feature space (Huben et al., 2024), but they still require a clean discovery signal. The central challenge is to construct a controlled internal contrast that holds task content fixed while placing the model in meaningfully different pre-generation states.

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.

Our key insight is that reasoning-related intervention targets can be discovered before reasoning is generated. For the same task content x, we compare a chat template input $T_{\text{chat}}(x)$ with a raw continuation input $T_{\text{raw}}(x)$ at their functionally aligned final prompt positions, immediately before decoding begins. Chat prepares an assistant response while Raw remains on a continuation trajectory; both may later produce reasoning traces, but their states already differ before generation.

We turn this contrast into an intervention target through feature screening and layer localization. SAE features are first ranked by their activation-frequency contrast between the matched Chat and Raw conditions. Prompt-component controls then remove responses that can be reproduced by isolated parts of the chat template, while layerwise Logits Lens analysis localizes the earliest consistent trajectory separation to layers 23 and 24, before the visible <think> entry emerges in upper layers. Restricting the controlled candidates to this localized range yields 23 intervention targets in Qwen3-8B.

Direct intervention reveals substantial causal leverage across three task families. Targeted SAE-guided suppression of the 23 features throughout generation over $2{,}187$ examples from GSM8K, CruxEval-O, and HumanEvalPlus reduces sample-weighted accuracy from 86.1% to 41.2%, a drop of 44.9 percentage points. On GSM8K, same-size, same-layer-allocation random null interventions reduce accuracy by only 4.4 points on average, compared with 44.2 points under the targeted intervention. On the code benchmarks, specificity relative to random ablation is task dependent, but targeted and random interventions produce distinct failure profiles. Visible chain-of-thought controls further show that disabling visible reasoning does not reproduce the collapse, while forcing visible reasoning entry after intervention does not restore performance. Reversing and scaling the same intervention operator produces structured and asymmetric behavioral changes.

These results identify a compact, intervenable feature-level target with broad causal influence across mathematical and code-related behavior. The discovery strategy connects a content-matched pre-generation contrast to causal tests across tasks, providing a concrete route from internal state differences to behavioral consequences. It also supplies a target for future comparisons across training checkpoints and code-data mixtures.

Our contributions are threefold:

- **A temporal principle for discovering intervention targets.** We introduce a content-matched contrast between functionally aligned Chat and Raw final prompt states to identify pre-generation features for subsequent causal testing.
- **A compact feature-level target with broad causal influence.** Controlled feature screening and layer localization yield 23 SAE features in layers 23 and 24 for which joint SAE-guided intervention reduces aggregate accuracy by 44.9 percentage points across mathematical reasoning, code output prediction, and functional code generation.
- **A causal characterization of the identified target.** Matched random nulls, failure analysis, visible-CoT controls, and direction-and-strength sweeps of the same operator reveal structured and task-dependent effects, linking a common causal target to distinct patterns of behavioral failure.

2 Related Work and Positioning

Code as a training signal and a reasoning medium. Controlled studies have begun to isolate how code data affects capabilities beyond programming. Varying the pretraining mixture, the stage at which code is introduced, or the amount of continued code training has revealed effects on mathematics, structured compositional tasks, broader non-code evaluations, and entity-state tracking (Petty et al., 2025; Ma et al., 2024; Kim et al., 2024; Aryabumi et al., 2025). Although the gains are task- and mixture-dependent, this line of work establishes the code–reasoning connection at the behavioral level. A complementary literature uses code directly as a reasoning medium: PAL and Program of Thoughts ask language models to express intermediate computation as executable programs and delegate execution to an external interpreter (Gao et al., 2023; Chen et al., 2023). These approaches demonstrate the usefulness of program structure during inference, but do not identify what internal computation is learned from code exposure and subsequently reused when no program is generated.

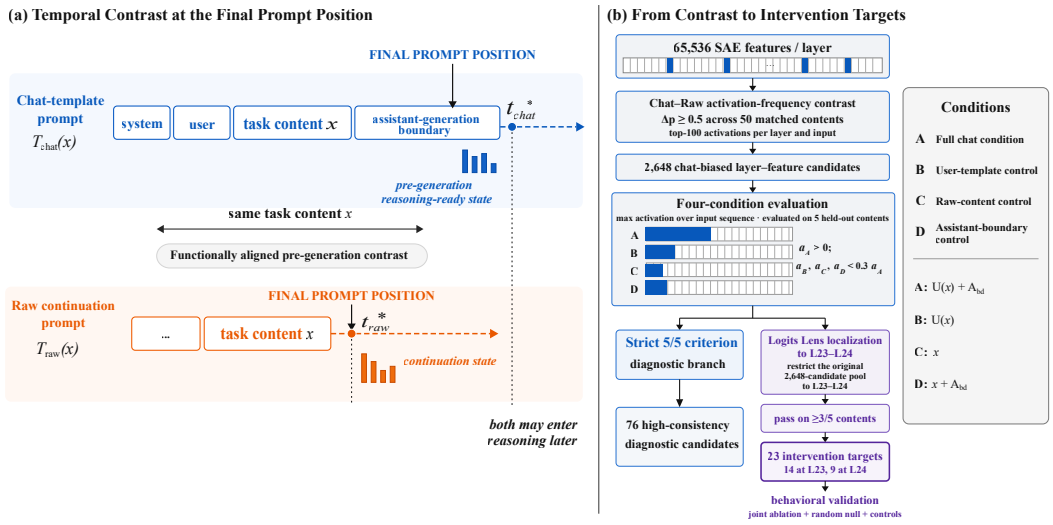

Figure 1: **Intervention-target discovery.** (a) Content-matched Chat and Raw inputs define a temporal contrast at the final pre-generation state. (b) Frequency ranking, prompt-component controls, and layer localization yield 23 targets in L23–L24.

Internal mechanisms of chain-of-thought reasoning. Chain-of-thought prompting improves multi-step task performance, while faithfulness studies show that the visible trace need not transparently reveal the computation determining the answer (Wei et al., 2022; Turpin et al., 2023; Lanham et al., 2023). Mechanistic analyses have begun to move below the output level. Dutta et al. (2024) identify parallel pathways and a functional transition across layers during CoT reasoning, while Cabannes et al. (2024) recover specialized iteration heads in a controlled setting. Sparse autoencoders provide a finer feature space for causal analysis (Huben et al., 2024; Marks et al., 2024). Most closely related to our work, Chen et al. (2026) patch SAE features between CoT and no-CoT runs in Pythia on GSM8K, and He et al. (2026) contrast direct and CoT prompting to identify features whose steering can elicit reasoning behavior. Li et al. (2025) extract SAE features from verbal and symbolic CoT components and steer residual states during generation.

Our positioning. We instead derive the discovery signal from content-matched Chat and Raw final prompt states, before either condition generates a reasoning trace. Prompt-component controls and layer localization turn this contrast into a compact target for full-generation causal intervention across mathematical and code-related tasks. The contribution is a discovery strategy that connects pre-generation state differences to cross-task behavioral effects, while providing a concrete target for investigating the broader code–reasoning connection.

3 From Temporal Contrast to Intervention Targets

We use Qwen3-8B (Yang et al., 2025) as our testbed. Its visible reasoning trajectory and available layerwise residual-stream SAE make it a tractable setting for tracing a pre-generation contrast to feature-level intervention.

3.1 A Temporal Contrast at the Final Prompt Position

Given the same task content x, we construct the two input contexts illustrated in Fig. 1 (a). $T_{\text{chat}}(x)$ places x inside the model's chat template and ends at the assistant-generation boundary, whereas $T_{\text{raw}}(x)$ presents the same content as a plain-text continuation without chat markers. The task content is therefore held fixed while the surrounding generation context changes.

Let $t^\star_{\text{chat}}$ and $t^\star_{\text{raw}}$ denote the final non-padding prompt positions of the two inputs. These positions need not correspond to the same token, sequence index, or prompt length. They are functionally

aligned as the final internal states immediately before generation begins. At layer ℓ, we compare

$$h_\ell^{\text{chat}}(x, t^\star_{\text{chat}}) \quad \text{and} \quad h_\ell^{\text{raw}}(x, t^\star_{\text{raw}}).$$

Here, *reasoning-ready* labels the Chat state preceding visible reasoning. The contrast identifies features associated with preparing the same task content for assistant generation; subsequent interventions test their influence on task performance.

3.2 Ranking and Filtering SAE Features

As summarized in Fig. 1(b), we first map the temporal state contrast into SAE feature space and identify differential responses that recur across matched inputs. We use `SAE-Res-Qwen3-8B-Base-W64K-L0_100` (Deng et al., 2026), a layerwise residual-stream SAE with 65,536 features per layer. For each of $N = 50$ matched task contents, we encode the residual state at the final non-padding prompt position under both $T_{\text{chat}}(x)$ and $T_{\text{raw}}(x)$. At each layer and for each input, we retain the top-100 feature activations and set the remainder to zero; a retained feature is treated as active when its activation is positive.

For feature j at layer ℓ, we estimate its empirical activation frequency under condition $c \in \{\text{chat}, \text{raw}\}$ as

$$\hat{p}^c_{\ell,j} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\Big[\tilde{f}^c_{\ell,j}\big(x_i, t^\star_{i,c}\big) > 0\Big],$$

where $\tilde{f}$ denotes the top-100-truncated SAE activation. We then define

$$\Delta p_{\ell,j} = \hat{p}^{\text{chat}}_{\ell,j} - \hat{p}^{\text{raw}}_{\ell,j}.$$

The ranking stage in Fig. 1(b) retains features with $\Delta p_{\ell,j} \geq 0.5$, yielding 2,648 Chat-biased layer-feature candidates. Using empirical activation frequency emphasizes differential responses that recur across matched contents rather than those dominated by a small number of unusually large activations.

The filtering stage in Fig. 1(b) then asks whether a Chat-biased response requires the complete chat context or can be reproduced by isolated prompt components. For each candidate, we evaluate the four matched conditions shown in the figure:

$$A = U(x) + A_{\text{bd}},\ B = U(x),\ C = x,\ D = x + A_{\text{bd}}.$$

where $U(x)$ denotes the user-templated content and A_{bd} the assistant-generation boundary. Let $a_A, \ldots, a_D$ denote the maximum sequence activation of the candidate under the four conditions. For a given content x, the candidate passes the control when

$$a_A > 0 \qquad \text{and} \qquad \max(a_B, a_C, a_D) < 0.3 a_A.$$

We evaluate this criterion on five held-out task contents. Requiring the criterion to hold for all five yields 76 high-consistency diagnostic candidates, establishing that a subset of the temporal feature signal consistently depends on the complete chat context. The final intervention set is constructed separately after layer localization, starting again from the original 2,648-candidate pool.

3.3 Localizing the Intervention Layers

We inspect the earliest observed trajectory separation with an untuned Logits Lens (Belrose et al., 2023), projecting layer states directly through the model's output head. For each of five matched contents, we inspect the top-5 projections at the Chat and Raw final prompt positions in every layer (Supplementary Appendix A).

Figure 2 reveals a consistent progression across all five contents. Projections in L0–L15 remain diffuse, with top-token probabilities below 10^{-3}. From L16 to L22, both trajectories sharpen but remain broadly similar. A sharp separation first appears at L23–L24: Raw begins to favor task-specific continuation tokens, whereas Chat shifts toward assistant-response tokens, with representative projected probabilities rising from approximately 0.1 to 0.95 and 0.01 to 0.93, respectively. The trajectories diverge further in L25–L33, where Raw increasingly favors task-specific answers while

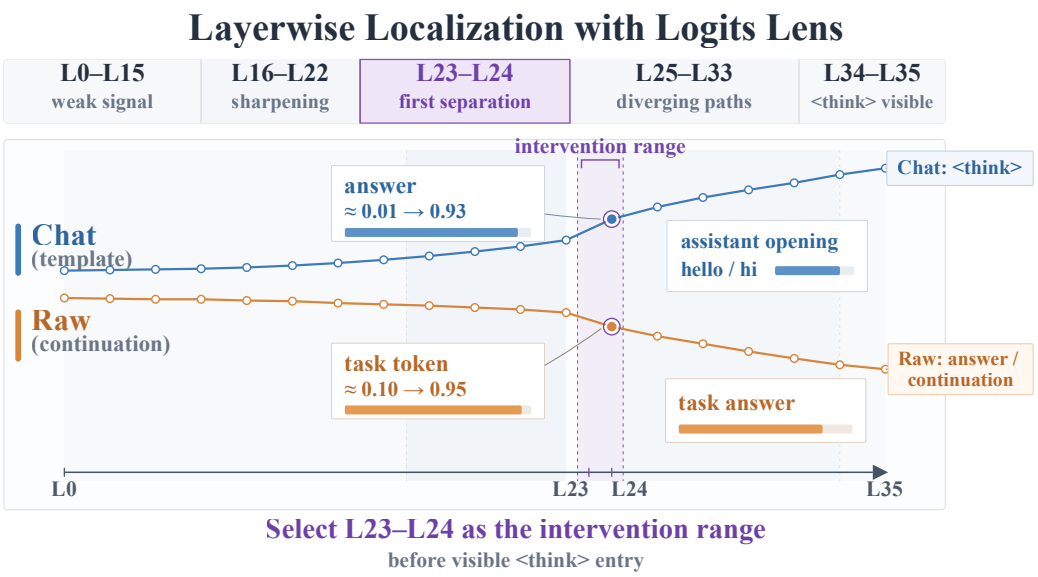

Figure 2: **Layerwise localization.** Across five matched contents, Chat and Raw first separate consistently at L23–L24, before visible <think> entry.

Chat favors assistant-style openings. Only in L34–L35 does the visible <think> token become dominant under Chat.

We therefore select L23–L24 as the intervention range. We then return to the original 2,648 Chat-biased candidates, restrict them to L23–L24, and apply the prompt-component control defined above. A feature is retained when it satisfies the control criterion on at least three of the five held-out contents, yielding 23 intervention targets: 14 in L23 and 9 in L24. The strict 5/5 set serves as a high-consistency diagnostic of the controlled feature signal, while the localized 3/5 criterion retains responses that recur across contents without requiring universal activation, yielding the final intervention set.

4 Causal Validation Design

We test three questions: whether joint intervention on the identified target changes behavior across tasks, how its effects compare with matched random perturbations, and which aspects of the generation trajectory are disrupted. Targeted suppression, random nulls, and behavioral controls address these questions in turn.

4.1 Intervention and Null Conditions

We evaluate the identified target set using an SAE-guided residual-stream intervention. For a target set S_ℓ at layer ℓ, the intervention is applied to the block-output residual at the current generation frontier. Let h_ℓ denote this residual and $\hat{d}_{\ell,j}$ the ℓ_2-normalized SAE decoder direction of feature j. We first encode h_ℓ with the layerwise SAE, retain its top-100 activations, set the activations of features in S_ℓ to zero, and decode the modified representation to obtain $h^{(S_\ell)}_{\ell,\mathrm{recon}}$. The intervened residual is

$$h'_\ell = h^{(S_\ell)}_{\ell,\mathrm{recon}} - \lambda \sum_{j \in S_\ell} \left(h_\ell^\top \hat{d}_{\ell,j} \right) \hat{d}_{\ell,j},$$

with

$$\hat{d}_{\ell,j} = \frac{W_{\mathrm{dec},\ell}[:, j]}{\|W_{\mathrm{dec},\ell}[:, j]\|_2}.$$

All directional coefficients are computed from the same original residual h_ℓ, avoiding sequential update dependence among features within a layer. The operator acts on the final prompt position

and on the newly generated token at every autoregressive decoding step, continuously perturbing the active generation trajectory rather than applying a one-shot edit.

Our targeted condition applies this operator jointly to all 23 identified features, with 14 targets in L23 and 9 in L24, using $\lambda = 3$. In the implementation, $\lambda > 0$ suppresses alignment with the selected decoder directions and $\lambda < 0$ reinforces it; $\lambda = 3$ therefore constitutes the primary over-suppression setting used for causal testing. The reported effects concern this joint operator on the selected set (Supplementary Appendix A).

We compare the targeted condition with an unmodified Baseline and four independently sampled random-null feature sets, generated using seeds $\{0, 1, 2, 3\}$. Each null uses the identical intervention operator, $\lambda = 3$, and the same 14-at-L23 plus 9-at-L24 allocation. Random features are sampled uniformly from the full 65,536-feature dictionary of each layer while excluding the 23 identified targets. The null matches set size, layer allocation, reconstruction procedure, operator strength, and application timing, while varying the selected feature identities. It is not activation-, frequency-, candidate-, or semantic-matched.

4.2 Benchmarks and Evaluation

We test whether the localized target exerts causal influence across three distinct task families: GSM8K (Cobbe et al., 2021) with 1,319 examples for mathematical reasoning, CruxEval-O (Gu et al., 2024) Adv with 704 examples for code-output prediction, and HumanEvalPlus (Liu et al., 2023) with 164 problems for functional code generation. Together they provide 2,187 evaluation instances spanning both mathematical and code-related behavior. We report benchmark accuracy and a sample-weighted aggregate across all examples.

Evaluation follows each benchmark's output structure. GSM8K predictions are scored by extracting a boxed answer when present and otherwise using an answer-cue and final-number fallback; extraction failures are counted as incorrect. CruxEval-O uses final-answer extraction and Python-value comparison under a condition-independent scoring audit (Supplementary Appendix A). Corrections recover explicitly committed answers without changing generated text or repairing reasoning; unresolved extractions count as incorrect. Think-marker closure is not a correctness criterion. HumanEvalPlus uses the standard all-tests-pass criterion after code extraction.

All generations use Qwen3-8B in bfloat16 with temperature 0.3, served through the same inference stack on an $8\times$H20 node. The generation cap is 2,000 tokens for the primary GSM8K Baseline and targeted conditions, 3,000 for CruxEval-O, and 8,192 for HumanEvalPlus. Random GSM8K runs use a 3,000-token cap; re-scoring them after truncation to 2,000 tokens changes mean accuracy by only approximately 0.8 percentage points, leaving the comparison unchanged.

4.3 Behavioral Controls and Diagnostics

Behavioral controls and failure diagnostics distinguish changes in task performance, visible chain-of-thought entry, and output degeneration.

We distinguish *CoT entry*, defined by the presence of an opening <think> marker, from *CoT closure*, which additionally requires a subsequent closing marker in the correct order. Two complementary controls probe the role of visible reasoning. **No-Think** disables the model's visible thinking mode without applying the feature intervention, testing whether removing overt CoT alone reproduces the targeted effect. **Force-Think** applies the targeted intervention but forces an opening <think> marker, testing whether restoring visible reasoning entry is sufficient to recover downstream behavior.

We further analyze intervention-induced failures rather than treating every incorrect output as equivalent. On HumanEvalPlus, outputs are separated into passing programs, extractable but test-failing programs, and outputs from which no executable program can be extracted. On GSM8K, incorrect outputs are categorized according to whether a final answer is absent and by recurrent degeneration patterns such as format collapse and repetitive loops. These diagnostics expose whether targeted and random perturbations fail through the same behavioral pathways.

Finally, we vary intervention direction and strength using $\lambda \in \{-3, -1, +1, +2, +3\}$. The same operator and target set are used throughout; negative values reinforce the selected directions and

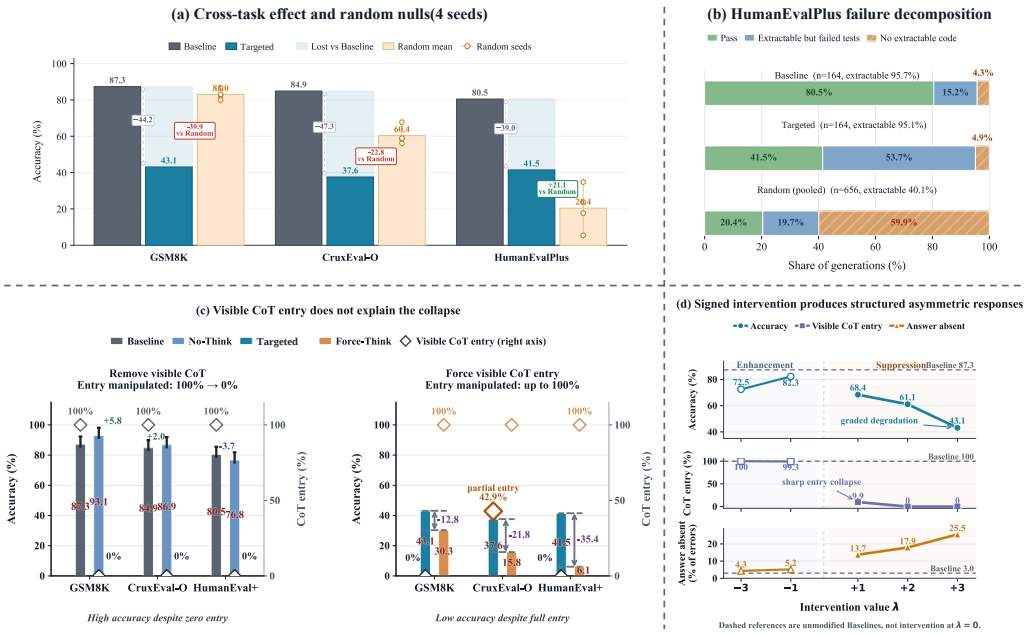

Figure 3: **Causal characterization of the identified intervention target.** (a) Targeted SAE-guided suppression degrades all tasks; four random nulls show strong separation on GSM8K, substantial separation on CruxEval-O, and reversed ordering on HumanEvalPlus. (b) On HumanEvalPlus, targeted suppression preserves extractability but breaks correctness, whereas random nulls often prevent code extraction. (c) Visible CoT entry neither explains nor rescues the collapse. (d) Signed strength sweeps yield graded, asymmetric changes in accuracy, CoT entry, and answer absence. Dashed lines mark unmodified Baselines.

positive values suppress them. The unmodified model serves as the Baseline and is not equivalent to invoking the intervention with $\lambda = 0$, since the latter retains top-100 SAE reconstruction with the target activations zeroed. This sweep tests whether the identified target behaves as a graded causal control surface rather than a binary on/off switch.

5 Causal Evaluation

Figure 3 characterizes joint SAE-guided intervention on the 23-feature target through cross-task effects, random nulls, failure profiles, visible-CoT controls, and signed interventions. Table 1 adds aggregate and trajectory-level diagnostics.

5.1 Broad Cross-Task Degradation

Targeted suppression produces a large and cross-task consistent collapse, reducing accuracy by 39.0**–**47.3 **points on every benchmark and by** 44.9 **points in aggregate.** As shown in Fig. 3(a), targeted SAE-guided suppression of the 23 intervention targets reduces accuracy from 87.3% to 43.1% on GSM8K, from 84.9% to 37.6% on CruxEval-O, and from 80.5% to 41.5% on HumanEvalPlus. Across all 2,187 evaluation examples, sample-weighted accuracy falls from 86.1% to 41.2%, a 44.9-percentage-point drop. The effect spans tasks with distinct output structures and evaluation criteria: joint intervention on the same compact target affects mathematical reasoning, code-output prediction, and functional code generation.

5.2 Specificity and Failure Divergence

Accuracy-level specificity is task dependent, but targeted and random interventions produce sharply different behavioral signatures. Figure 3(a) compares targeted suppression with four same-size, same-layer-allocation random null interventions using the same SAE-guided operator

Table 1: **Additional behavioral diagnostics beyond Fig. 3.** Cross-task controls report sample-weighted accuracy over all 2,187 examples; random-null ranges span four independently sampled null sets (4 seeds).

Diagnostic	Scope	Result (%)
Cross-task control effect		
No-Think accuracy	All 2,187	**89.9** (+3.8 vs. Base)
Force-Think accuracy	All 2,187	**23.8** (−17.4 vs. Targeted)
Random-null visible CoT entry		
Entry range	GSM8K	5.2–40.6
	CruxEval-O	95.5–100
	HumanEvalPlus	97–100
Force-Think trajectory stability		
Unclosed	GSM8K errors ($n = 920$)	**96.6**
Repetitive	GSM8K errors ($n = 920$)	**39.9**

and intervention strength. On GSM8K, the random nulls retain $83.0 \pm 2.4\%$ accuracy on average, compared with 43.1% under targeted suppression, leaving a 39.9-point gap. CruxEval-O shows the same ordering with a 22.8-point gap between the random mean of $60.4 \pm 4.4\%$ and the targeted result of 37.6%. HumanEvalPlus reverses this ordering: random nulls reach $20.4 \pm 10.6\%$, below the targeted result of 41.5%. All four independently sampled null sets preserve this task-wise ordering. The cross-task result is therefore a common causal vulnerability with task-dependent specificity.

Figure 3(b) shows why scalar accuracy alone is insufficient to characterize this difference. Under targeted suppression, code extractability remains 95.1%, almost unchanged from the 95.7% Baseline, even though only 41.5% of generations pass the test suite. In contrast, random nulls reduce extractability to 40.1%. More specifically, 53.7% of targeted generations remain extractable but fail the tests, whereas 59.9% of random-null generations yield no extractable code. The two interventions thus expose distinct failure modes: targeted suppression primarily breaks functional correctness, while random perturbations frequently prevent code extraction.

The random nulls further expose a behavioral dissociation not shown in the main figure. As reported in Table 1, visible CoT entry ranges from only 5.2% to 40.6% across the four random sets on GSM8K, where the random-null mean accuracy remains 83.0%, but stays at 95.5%–100% on CruxEval-O and 97%–100% on HumanEvalPlus despite substantial performance degradation. Visible reasoning entry therefore does not track performance degradation in a task-independent manner.

5.3 Visible CoT Entry Does Not Explain the Collapse

Visible CoT entry is neither required for high performance nor sufficient to rescue the targeted collapse. Figure 3(c) tests this explanation in both directions. No-Think eliminates visible CoT entry on all three benchmarks, yet accuracy remains 93.1% on GSM8K, 86.9% on CruxEval-O, and 76.8% on HumanEvalPlus. Conversely, Force-Think forces the opening <think> marker after targeted suppression, yielding 100% measured CoT entry on all three benchmarks, but accuracy falls further to 30.3%, 15.8%, and 6.1%, respectively. The same pattern holds in aggregate: Table 1 shows 89.9% sample-weighted accuracy under No-Think, compared with 86.1% for the unmodified Baseline, whereas Force-Think reaches only 23.8%, below the 41.2% targeted condition.

Forced entry also fails to restore a stable subsequent trajectory. On GSM8K, 889 of the 920 Force-Think errors (96.6%) fail to close the visible reasoning block, while 367 (39.9%) exhibit repetitive behavior; these diagnostics are non-exclusive (Table 1). These results distinguish entering a visible reasoning trace from sustaining it to completion. Together with the No-Think and random-null results, they show that the targeted effect cannot be reduced to a simple switch controlling whether the model emits an opening <think> marker.

5.4 Direction and Strength Reveal Asymmetric Control

The same target exhibits signed, strength-dependent, and asymmetric behavioral responses. Figure 3(d) varies the same intervention operator over $\lambda \in \{-3, -1, +1, +2, +3\}$, where negative values reinforce alignment with the selected decoder directions and positive values suppress it. The dashed references denote the unmodified model and are not intervention runs at $\lambda = 0$.

Increasing suppression from $\lambda = +1$ to $+3$ reduces GSM8K accuracy from 68.4% to 61.1% and then 43.1%. Over the same settings, visible CoT entry falls from 9.9% to 0%, while the share of errors without a final answer rises from 13.7% to 17.9% and 25.5%. Performance, visible reasoning entry, and failure composition therefore evolve along related but non-identical response profiles.

The negative-λ branch is not a mirror image of suppression. At $\lambda = -1$, accuracy remains relatively close to the 87.3% Baseline at 82.3%, with 99.3% visible CoT entry and only 5.2% answer absence among errors. Stronger negative intervention at $\lambda = -3$, however, lowers accuracy to 72.5% despite preserving 100% visible CoT entry and a similarly low 4.3% answer-absence rate. Performance can therefore deteriorate even when visible reasoning entry is preserved and answer absence among errors remains low. The same target thus exhibits graded, sign-sensitive, and asymmetric causal responses rather than a binary on/off effect.

6 Discussion

From pre-generation contrast to causal intervention. A contrast taken before the first visible reasoning token identifies features whose joint manipulation throughout generation strongly affects later computation. HumanEvalPlus makes the behavioral distinction clear: targeted intervention largely preserves extractable code while often breaking functional correctness. Structured output can therefore survive even when the computation expressed through it fails. Together, the discovery and intervention results connect pre-generation organization to a common causal target across tasks.

Reasoning requires more than starting a trace. No-Think preserves high task performance, whereas Force-Think restores visible entry without rescuing the targeted collapse. This points to sustaining computation as a central question: carrying intermediate results forward, avoiding repetition, and reaching an answer. Autoregressive generation makes this process path dependent, motivating methods that sample or search alternative reasoning paths (Wang et al., 2023; Yao et al., 2023). A natural next test is to relate target-feature activity to token-level entropy and later trajectory failure, asking whether the target helps manage uncertainty at such decision points.

Why might code help? If successful reasoning depends on keeping a multi-step computation on track, code offers a natural source of training for that ability. One account emphasizes reusable primitives such as iteration, recursion, and conditional branching; another is that code provides dense practice in sustaining computation across dependent steps. Syntax constrains local choices, variable bindings preserve state, control flow links present actions to future execution, and program semantics test global coherence. The cross-task effects and separation of extractability from correctness motivate a concrete hypothesis: code exposure may strengthen the organization of multi-step computation. The 23-feature target gives this hypothesis an internal address: when does it emerge, how does code exposure change it, and do those changes precede reasoning gains?

Limitations. The current evidence concerns joint intervention in one model with one SAE. Establishing reasoning specificity and discovery stability requires broader task controls and sensitivity analyses. We do not yet trace how code exposure forms or reshapes the target; closing this training-time link is the key step toward a complete account of code-to-reasoning transfer.

7 Conclusion

We introduce a temporal contrast that identifies a compact 23-feature SAE intervention target in Qwen3-8B before generation begins. Joint SAE-guided intervention throughout generation disrupts mathematical and code-related tasks, with failure profiles that distinguish functional correctness from output form and visible reasoning entry. These findings establish pre-generation contrast as a discovery strategy for locating cross-task causal targets and provide a concrete basis for studying how training shapes them.

AI USE STATEMENT

We used generative AI tools solely to improve the grammar, wording, and readability of this manuscript. All AI-assisted revisions were reviewed by the authors to ensure that they preserved the intended meaning and technical accuracy. The authors take full responsibility for the final content of this manuscript.

REFERENCES

Viraat Aryabumi, Yixuan Su, Raymond Ma, Adrien Morisot, Ivan Zhang, Acyr Locatelli, Marzieh Fadaee, Ahmet Üstün, and Sara Hooker. To code, or not to code? exploring impact of code in pre-training. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=zSfeN1uAcx.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023. doi: 10.48550/arXiv.2303.08112. URL https://arxiv.org/abs/2303.08112.

Vivien Cabannes, Charles Arnal, Wassim Bouaziz, Alice Yang, Francois Charton, and Julia Kempe. Iteration head: A mechanistic study of chain-of-thought. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3463. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/c50f8180ef34060ec59b75d6e1220f7a-Abstract-Conference.html.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Xi Chen, Aske Plaat, and Niki van Stein. How does chain of thought think? mechanistic interpretability of chain-of-thought reasoning with sparse autoencoding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30297–30305, 2026. doi: 10.1609/aaai.v40i36.40281. URL https://ojs.aaai.org/index.php/AAAI/article/view/40281.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. doi: 10.48550/arXiv.2110.14168. URL https://arxiv.org/abs/2110.14168.

Boyi Deng, Xu Wang, Yaoning Wang, Yu Wan, Yubo Ma, Baosong Yang, Haoran Wei, Jialong Tang, Huan Lin, Ruize Gao, Tianhao Li, Qian Cao, Xuancheng Ren, Xiaodong Deng, An Yang, Fei Huang, Dayiheng Liu, and Jingren Zhou. Qwen-Scope: Turning sparse features into development tools for large language models. *arXiv preprint arXiv:2605.11887*, 2026. doi: 10.48550/arXiv.2605.11887. URL https://arxiv.org/abs/2605.11887.

Subhabrata Dutta, Joykirat Singh, Soumen Chakrabarti, and Tanmoy Chakraborty. How to think step-by-step: A mechanistic understanding of chain-of-thought reasoning. *arXiv preprint arXiv:2402.18312*, 2024. doi: 10.48550/arXiv.2402.18312. URL https://arxiv.org/abs/2402.18312.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Alex Gu, Baptiste Roziere, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida Wang. CRUXEval: A benchmark for code reasoning, understanding and execution. In

Proceedings of the 41st International Conference on Machine Learning, volume 235 of *Proceedings of Machine Learning Research*, pp. 16568–16621. PMLR, 2024. URL https://proceedings.mlr.press/v235/gu24c.html.

Zhenghao He, Guangzhi Xiong, Bohan Liu, Sanchit Sinha, and Aidong Zhang. Reasoning beyond chain-of-thought: A latent computational mode in large language models. *arXiv preprint arXiv:2601.08058*, 2026. doi: 10.48550/arXiv.2601.08058. URL https://arxiv.org/abs/2601.08058.

Robert Huben, Hoagy Cunningham, Logan Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

Najoung Kim, Sebastian Schuster, and Shubham Toshniwal. Code pretraining improves entity tracking abilities of language models. *arXiv preprint arXiv:2405.21068*, 2024. doi: 10.48550/arXiv.2405.21068. URL https://arxiv.org/abs/2405.21068.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, Saurav Kadavath, Shannon Yang, Thomas Henighan, Timothy Maxwell, Timothy Telleen-Lawton, Tristan Hume, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023. doi: 10.48550/arXiv.2307.13702. URL https://arxiv.org/abs/2307.13702.

Zihao Li, Xu Wang, Yuzhe Yang, Ziyu Yao, Haoyi Xiong, and Mengnan Du. Feature extraction and steering for enhanced chain-of-thought reasoning in language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10893–10913. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.552. URL https://aclanthology.org/2025.emnlp-main.552/.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Yingwei Ma, Yue Liu, Yue Yu, Yuanliang Zhang, Yu Jiang, Changjian Wang, and Shanshan Li. At which training stage does code data help LLMs reasoning? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KIPJKST4gw.

Samuel Marks, Can Rager, Eric J. Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. *arXiv preprint arXiv:2403.19647*, 2024. doi: 10.48550/arXiv.2403.19647. URL https://arxiv.org/abs/2403.19647.

Jackson Petty, Sjoerd van Steenkiste, and Tal Linzen. How does code pretraining affect language model task performance? *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=pxxmUKKgel.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Advances in Neural Information Processing Systems*, volume 36, pp. 74952–74965, 2023. URL https://openreview.net/forum?id=bzs4uPLXvi.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

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

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023. URL https://openreview.net/forum?id=5Xc1ecxO1h.

A Implementation Details

Prompt serialization. The discovery walkthrough in the accompanying code gives the following explicit Chat–Raw pair, where {x} denotes identical task content and \n denotes a newline:

```
Chat: <|im_start|>user\n{x}<|im_end|>\n<|im_start|>assistant\n
Raw:  {x}
```

The comparison endpoint tokenizes the supplied strings directly. Thus the Chat string includes the assistant-generation boundary, whereas the Raw string contains only the task content. These are serialization examples, not the archived discovery samples. The main text defines the four component controls. The strict 5/5 diagnostic branch and the localized 3/5 target-selection branch both start from the frequency-screened candidate pool; the 23 targets are not obtained by simply taking a subset of the 76 diagnostic candidates.

Layerwise vocabulary projection. The `/logits_lens` endpoint in `qwen3_server.py` takes the hidden-state tensor returned at index $\ell + 1$, selects the final prompt position, and applies the model's output embedding head followed by a vocabulary softmax. It returns the top-five token projections for the analysis in the main text. There is no learned lens translator or additional normalization in this projection routine. Layer selection follows inspection of the Chat–Raw token-trajectory separation across the five contents; the projected probabilities are diagnostic outputs, not a separately calibrated measure of reasoning readiness.

Joint intervention semantics. The implementation `_intervene_sae` first retains the top-100 encoder activations and zeros the selected target activations. It then decodes this representation and adds the sum of signed decoder-direction updates. Each direction has unit Euclidean norm, and every projection coefficient is computed from the same original residual. The decoder directions are not orthogonalized, so the sum of their projections is not an orthogonal projection onto their span. Accordingly, the experiments measure the effect of the specified joint operator and do not decompose it into independent contributions of the 23 features.

At $\lambda = 0$, the directional update vanishes but target-zeroed reconstruction remains. This differs both from the unmodified model and from reconstruction with all retained activations intact. Positive and negative values change the sign of the directional update; they do not make the two full interventions algebraic inverses. The generation hook applies the operator at the current final position on each forward pass, covering the final prompt position and subsequent autoregressive steps.

Exact target configuration. The target IDs below agree across the benchmark runners and the random-feature sampling utility. Layer indices are zero-based.

L23 (14 features): 32698, 58619, 6303, 9238, 56137, 45328, 43536, 58345, 55669, 11636, 59607, 29193, 39269, 17368.

L24 (9 features): 5168, 650, 6643, 3470, 51476, 57828, 62131, 46830, 38617.

The full-dictionary null configurations are specified in `configs/true_random_features.json` and generated by `prepare_true_random_features.py`. The four reported null sets

use seeds 0–3, with 14 features at L23 and nine at L24, excluding the target IDs. Variability across these sets concerns random feature selection, rather than repeated generation seeds for a fixed intervention.

Code organization. The accompanying artifact contains the intervention server, target and null configurations, benchmark runners, and verification scripts. The principal components are `qwen3_server.py` for state extraction and intervention; `run_gsm8k_C.py`, `run_cruxeval.py`, and the HumanEval runners for evaluation; and `verification/` for checks against generation records. In the CruxEval-O Adv runner, the input contains the function, its input, and supplied potentially erroneous reasoning; the model is asked to return the correct execution result. The scripts specify extraction and parsing procedures.

CruxEval-O scoring audit. The reported CruxEval-O Adv scores use a common post-generation audit across Baseline, the targeted intervention, the four random-null sets, No-Think, and Force-Think. The audit operates on the saved generations and preserves the generated text. It separates recovery of the model's explicitly committed final answer from comparison with the reference output.

Final-answer extraction and manual extraction review are performed without access to the reference answer or experimental condition. We identify the last explicit answer commitment, including a clearly stated self-correction. Think markers assist segmentation but are not correctness criteria: an unclosed think block does not invalidate an otherwise explicit final answer, and an intermediate value is not credited merely because it matches the reference. Conflicting answers without a clear final commitment remain unresolved.

Extraction repairs are restricted to meaning-preserving recovery of the submitted value, including complete boxed expressions with nested delimiters, code-fence removal, and intact decimal or container literals. Parsing success is tracked separately from the parsed value, so a valid `None` is distinguishable from failure. Parsed predictions are compared using the task's Python output-equality semantics, without an additional numerical tolerance. The audit does not execute model-generated reasoning, evaluate unfinished calculations on the model's behalf, or repair an incorrect answer. Extraction repairs and alignment of scoring behavior across runners are recorded separately.

The same rules apply to all comparison conditions and cover both initially correct and initially incorrect outputs, allowing score changes in either direction. Original and audited decisions are retained by sample ID, together with the recovered answer span and the reason for each change; audited totals are computed from these item-level decisions. Unresolved answers count as incorrect and remain in the denominator of 704 examples per condition. The released runners provide the initial extraction and scoring paths.